

Part II

Self-Evolution in Intelligent Agents

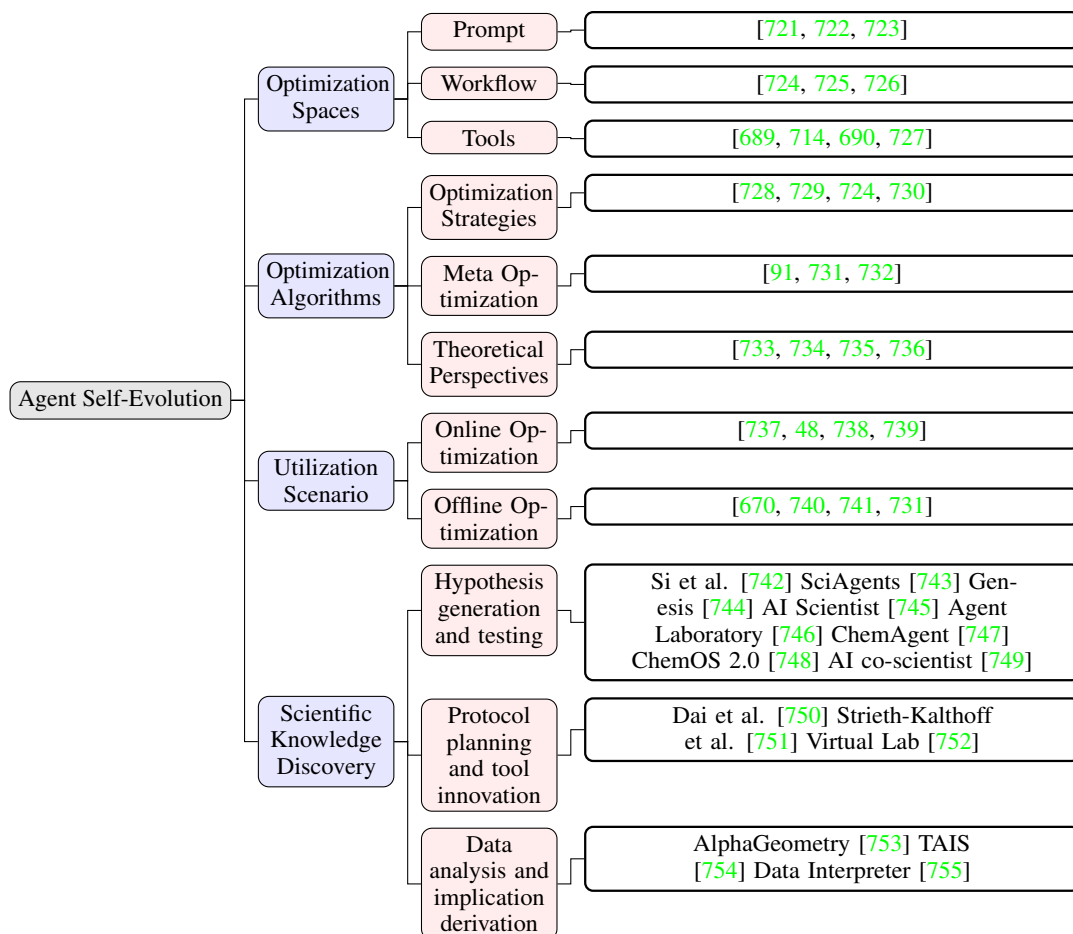


Figure 8.6: Structures of self-evolution in LLM agents.

In the history of machine learning research, manually designed AI systems have gradually been replaced by more efficient, learned solutions [756]. For instance, before the advent of deep learning, features were typically handcrafted by experts [757, 758], but these were eventually superseded by features extracted through neural networks. As neural networks have become increasingly complex, various techniques for automated design—such as neural architecture search—have emerged, further replacing the need for manually designed network structures [759]. Similarly, Agentic systems initially relied heavily on manual design, with behavior rules and decision-making strategies explicitly crafted by developers. Although full automation of agent self-evolution has not yet been achieved, it is anticipated and deemed necessary for future progress. A successful precedent for such automation can already be seen in automated machine learning (AutoML) [712, 760, 761, 762, 204] which has automated various components of traditional machine learning pipelines. In particular, AutoML streamlines the selection and configuration of machine learning algorithm pipelines while incorporating advanced techniques for hyperparameter optimization [763, 764, 765, 766, 767]. Among the most notable applications of AutoML is NAS [768, 769], which automates the design of neural network architectures to enhance model performance. Drawing inspiration from this successful transition towards automation in traditional machine learning, we propose extending similar principles to the domain of agentic AI systems.

A key counterintuitive issue in much of current agent research is that, while the ultimate goal of developing or improving agentic AI systems is to automate human efforts, the process of creating these systems remains, for the time being, beyond the reach of full automation. Therefore, we argue that all manually designed agentic AI systems will eventually be replaced by learnable and self-evolving systems, which could ultimately place the development and improvement of agentic AI into an autonomous, self-sustaining loop. Enabling self-evolution mechanism in LLM agents has the following benefits:

1. **Scalability:** While LLM-based agents have demonstrated remarkable performance, their improvement still heavily depends on the underlying LLMs. However, upgrading these models is costly, and scaling performance through the inclusion of additional real-world data requires extensive retraining on large datasets, which

poses significant resource constraints. Self-evolving agentic systems, in contrast, can optimize agent behavior without necessitating modifications to the underlying LLMs, offering a more efficient and scalable solution.

2. **Reduction in Labor Costs:** Manually designing agentic systems is a complex and labor-intensive process that requires developers to engage deeply with intricate technical details. Traditional methods often involve building these systems from scratch, demanding significant expertise and effort. By contrast, self-evolving agentic systems can automate much of this process, significantly reducing the need for manual intervention and lowering development costs.
3. **Aligned with Natural Intelligence Development:** Just as humans continuously improve themselves through learning and adaptation, equipping LLM agents with self-improvement capabilities is a necessary step toward the development of truly autonomous agents. This enables them to refine their performance, adapt to new challenges, and evolve without direct human intervention.

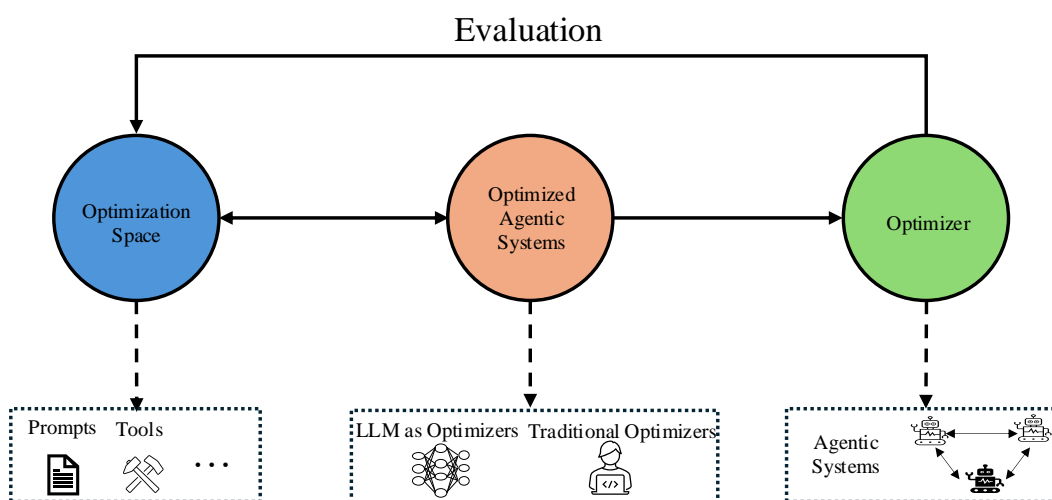


Figure 8.7: An illustration of key concepts discussed in this section, including optimization spaces, the optimizer, and the optimizing objective. The optimizer iteratively refines components within the optimization spaces to enhance agentic systems until a satisfactory outcome is achieved, thereby achieving self-improvement in the LLM agent systems.

To achieve the goal of automating human efforts, numerous studies have proposed leveraging LLMs as the driving engine to enable self-evolution in agentic systems. In particular, LLMs provide an efficient alternative to traditional optimization methods, such as gradient-based [770] and reinforcement learning-based approaches [771]. They extend the optimization space from numerical values to more diverse domains, with natural language serving as a universal bridge. An LLM is capable of optimizing complex, heterogeneous parameters, such as instructions [732] and tool implementations [772], and can operate across a range of LLMs, including both open-source and closed-source models. A notable example of this approach is AFLOW [773], which automates the generation and optimization of entire agentic system workflows. This system employs Monte Carlo Tree Search to leverage the comprehensive capabilities of LLMs. In this framework, traditionally handcrafted agentic systems are replaced by algorithmically generated ones, marking a kind of paradigm shift. Additionally, a growing body of research explores similar methodologies, further advancing the field.

This part is structured as follows: First, we introduce various optimization spaces explored in recent research on agentic systems, including prompts, tools, and workflows. In the subsequent section, we review optimization algorithms, discussing both traditional optimization paradigms and meta-optimization, where the optimization process also affects the underlying optimization algorithms themselves. We then explore the self-evolution scenarios, categorizing them into two types: online optimization and offline optimization. Following this, we discuss the application of large language model (LLM) agent self-improvement techniques, particularly in knowledge discovery within the AI-for-science domain. Finally, we discuss the security concerns associated with agent self-evolution technologies.

Chapter 9

Optimization Spaces and Dimensions for Self-evolution

The optimization of autonomous agents represents a complex challenge that encompasses multiple levels of abstraction. In this section, we first establish prompt optimization as the foundational layer, upon which three distinct branches of optimization emerge: agentic workflow optimization, tool optimization, and comprehensively autonomous agent optimization.

9.1 Overview of Agent Optimization

Existing LLM-based agent optimization can be conceptualized in terms of a two-tiered architecture. At the foundation lies *prompt optimization*, which focuses on enhancing the basic interaction patterns of Language Model nodes. Building upon this foundation, three parallel branches emerge: i) *workflow-level optimization*, which focuses on the coordination and interaction patterns between multiple LLM nodes; ii) *tool optimization*, where agents evolve by developing and improving tools to adapt to new tasks and leverage past data; and iii) *comprehensive autonomous agent optimization*, which aims at the holistic enhancement of agent capabilities by considering multiple dimensions.

Similarly to optimization paradigms in AutoML, agent optimization can be categorized as either single-objective or multi-objective. Contemporary agent optimization primarily centers on three canonical metrics: performance, inference cost, and latency. Performance measures the effectiveness of the agent in completing its assigned tasks, while inference cost quantifies the computational resources required for agent operation. Latency represents the time taken for the agent to respond and complete tasks. These objectives can vary depending on the specific optimization modality. For instance, in prompt-level optimization, additional constraints such as prompt length may become relevant objectives. This multi-faceted nature of optimization objectives reflects the complexity of agent systems and the need to balance multiple competing requirements.

9.2 Prompt Optimization

Prompt optimization plays the most critical role in LLM-based agent optimization. When optimizing an agent, beyond model-level optimizations, task-specific or model-specific prompt optimization directly impacts the agent’s performance, latency, and cost. Given a task $T = (Q, G_t)$, where Q denotes the input query and G_t represents the optional ground truth, the objective of prompt optimization is to generate a task-specific prompt P_t^* that maximizes performance:

$$P^* = \arg \max_{P \in \mathcal{P}} \mathbb{E}_{T \sim \mathcal{D}} [\phi_{\text{eval}}(\phi_{\text{exe}}(Q, P), T)] \quad (9.1)$$

where \mathcal{P} represents the space of possible prompts, ϕ_{exe} denotes the execution function, and ϕ_{eval} represents the evaluation function. This optimization is typically implemented through three fundamental functions: ϕ_{opt} , ϕ_{exe} , and ϕ_{eval} . The Optimize function ϕ_{opt} refines existing prompts based on optimization signals, the Execute function ϕ_{exe} invokes the current prompt to obtain output O , and the Evaluation function ϕ_{eval} assesses current outputs to generate evaluation signals S_{eval} and optimization signals S_{opt} . The evaluation signals are used to select effective prompts, while the optimization signals assist the Optimize function in performing optimization.

9.2.1 Evaluation Functions

At the core of prompt optimization lies the evaluation function ϕ_{eval} , which serves as the cornerstone for deriving optimization signals and guiding the evolutionary trajectory of prompts. This function orchestrates a sophisticated interplay between evaluation sources, methodologies, and signal generation, establishing a feedback loop that drives continuous improvement. The evaluation function ϕ_{eval} processes evaluation sources as input, and employs various evaluation methods to generate different types of signals, which subsequently guide the optimization process. Here, we define the dimensions of sources, methods, and signal types to establish the foundation for prompt optimization.

Evaluation Sources Evaluation sources primarily consist of LLM Generated Output G_{llm} and task-specific Ground Truth G_t . Existing works such as [730, 774, 728, 775, 732, 300] predominantly leverage comparisons between G_{llm} and G_t as evaluation sources. Some approaches [776, 721, 777] utilize only G_{llm} as the evaluation source. For instance, PROMST [721] assesses prompt effectiveness by comparing G_{llm} against human-crafted rules; SPO [778] employs pairwise comparisons of outputs from different prompts to determine relative effectiveness.

Evaluation Methods Evaluation Methods can be broadly categorized into three approaches: *benchmark-based evaluation*, *LLM-as-a-Judge*, and *human feedback*. *Benchmark-based evaluation* remains the most prevalent method in prompt optimization [730, 774, 721, 732, 300]. This approach relies on predefined metrics or rules to provide numerical feedback as evaluation signals. While it offers an automated evaluation process, its effectiveness ultimately depends on how well the benchmark design aligns with human preferences.

The introduction of *LLM-as-a-Judge* represents a significant advancement in automated evaluation and preference alignment. Leveraging LLMs’ inherent alignment with human preferences and carefully designed judging criteria, this approach [589] can assess task completion quality based on task descriptions and prompt outputs G_{llm} , providing reflective textual gradient feedback. Notable implementations include ProteGi [779], TextGrad [728], Semantic Search [775] and Revolve [780]. Furthermore, LLM-as-a-judge enables comparative evaluation between ground truth G_t and output G_{llm} with specific scoring mechanisms [724]. The effectiveness of this method hinges on both the design of judger prompts and the underlying model’s alignment with human preferences. As a specialized extension, *Agent-as-a-Judge* [781] refines this paradigm by employing dedicated agents for providing process evaluation on complex tasks, while maintaining high alignment with human preferences at significantly reduced evaluation costs.

Human feedback represents the highest level of intelligence integration in the evaluation process. As humans remain the ultimate arbiters of prompt effectiveness, direct human feedback can rapidly and substantially improve prompt quality. However, this approach introduces significant resource overhead. APOHF [777] demonstrates that incorporating human feedback can achieve robust prompt optimization with minimal computational resources, particularly excelling in open-ended tasks such as user instructions, prompt optimization for text-to-image generative models, and creative writing. Nevertheless, the requirement for human intervention somewhat contradicts the goal of automated evolution.

Signal Types Feedback generated by evaluation methods manifests in three distinct forms, each serving different optimization needs. *Numerical feedback* [730, 774, 721, 732, 300] quantifies performance through scalar metrics, compatible with rules, ground truth, human assessment, and LLM judgments. While widely applicable, this approach requires substantial samples for statistical reliability, potentially overlooking instance-specific details that could guide optimization. *Textual feedback* [728, 775, 780] provides detailed, instance-specific guidance through analysis and concrete suggestions. This sophisticated approach requires intelligent participation, either from human experts or advanced language models, enabling targeted improvements in prompt design through explicit recommendations. However, its reliance on sophisticated intelligence sources impacts its scalability. *Ranking feedback* [778] establishes relative quality ordering through either comprehensive ranking or pairwise comparisons. This approach uniquely circumvents the need for absolute quality measures or predefined criteria, requiring only preference judgments. It proves particularly valuable when absolute metrics are difficult to define or when optimization primarily concerns relative improvements.

9.2.2 Optimization Functions

The design of optimization functions is crucial in determining the quality of generated prompts in each iteration of prompt optimization. Through effective signal guidance, prompt self-evolution can achieve faster convergence. Current optimization approaches primarily rely on two types of signals: *evaluation signals* S_{eval} that identify the most effective existing prompts, and *optimization signals* S_{opt} that provide detailed guidance for improvements.

Optimize via Evaluation Signals When optimizing with evaluation signals, the process begins by selecting the most effective prompts based on ϕ_{eval} assessments. Rather than directly learning from past errors, some methods adopt

heuristic exploration and optimization strategies. SPO [778] iteratively refines prompts based on the outputs of current best-performing ones, leveraging the language model’s inherent ability to align with task requirements. Similarly, Evoprompt [723] employs evolutionary algorithms with LLMs serving as evolution operators for heuristic prompt combination. PromptBreeder [732] advances this approach further by comparing score variations between mutated prompts while simultaneously modifying both meta-prompts and prompts through the LLM’s inherent capabilities.

Optimize via Optimization Signals While optimization methods based solely on evaluation signals require extensive search to find optimal solutions in vast search spaces through trial and error, an alternative approach leverages explicit optimization signals to guide the optimization direction and improve efficiency. Existing methods demonstrate various ways to utilize these optimization signals. OPRO [730] extracts common patterns from high-performing prompt solutions to guide subsequent optimization steps. ProTegi [779] employs language models to analyze failure cases and predict error causes, using these insights as optimization guidance. TextGrad [728] extends this approach further by transforming prompt reflections into “textual gradients”, applying this guidance across multiple prompts within agentic systems. Revolve [780] further enhances optimization by simulating second-order optimization, extending previous first-order feedback mechanisms to model the evolving relationship between consecutive prompts and responses. This allows the system to adjust based on how previous gradients change, avoiding stagnation in suboptimal patterns and enabling more informed, long-term improvements in complex task performance.

9.2.3 Evaluation Metrics

The effectiveness of prompt optimization methods can be evaluated across multiple dimensions. *Performance metrics* [782, 778, 730] for Close Tasks serve as the most direct indicators of a prompt’s inherent performance, encompassing measures such as pass@1, accuracy, F1 score, and ROUGE-L. These metrics enable researchers to assess the stability, effectiveness, and convergence rate of prompt optimization processes. Another crucial dimension is *Efficiency metrics* [778]. While some prompt optimization approaches achieve outstanding results, they often demand substantial computational resources, larger sample sizes, and extensive datasets. In contrast, other methods achieve moderate results with lower resource requirements, highlighting the trade-offs between performance and efficiency in agent evolution. The third dimension focuses on qualitative metrics that assess specific aspects of agent behavior: consistency [776] measures output stability across multiple runs, fairness [783] evaluates the ability to mitigate the language model’s inherent biases, and confidence [784, 785] quantifies the agent’s certainty in its predictions. When these behavioral aspects are treated as distinct objectives, prompt optimization frameworks provide corresponding metrics for evaluation.

9.3 Workflow Optimization

While prompt-level optimization has shown promising results in enhancing individual LLM capabilities, modern AI systems often require the coordination of multiple LLM components to tackle complex tasks. This necessitates a more comprehensive optimization domain—the agentic workflow space. At its core, an agentic workflow consists of LLM-invoking nodes, where each node represents a specialized LLM component designed for specific sub-tasks within the larger system.

Although this architecture bears similarities to multi-agent systems, it is important to distinguish agentic workflows from fully autonomous multi-agent scenarios. In agentic workflows, nodes operate under predetermined protocols and optimization objectives, rather than exhibiting autonomous decision-making capabilities. Many prominent systems, such as MetaGPT [626] AlphaCodium [786] can be categorized under this framework. Moreover, agentic workflows can serve as executable components within larger autonomous agent systems, making their optimization crucial for advancing both specialized task completion and general agent capabilities.

Following the formalization proposed by GPTSwarm [651] and AFLOW [773], this section first establishes a formal definition of agentic workflows and their optimization objectives. We then examine the core components of agentic workflows—nodes and edges—analyzing their respective search spaces and discussing existing representation approaches in the literature.

9.3.1 Workflow Formulation

An agentic workflow \mathcal{K} can be formally represented as:

$$\mathcal{K} = \{(N, E) | N \in \mathcal{N}, E \in \mathcal{E}\} \quad (9.2)$$

where $\mathcal{N} = \{N(M, \tau, P, F) | M \in \mathcal{M}, \tau \in [0, 1], P \in \mathcal{P}, F \in \mathcal{F}\}$ represents the set of LLM-invoking nodes, with M , τ , \mathcal{P} , and \mathcal{F} denoting the available language models, temperature parameter, prompt space, and output format space respectively. E indicates the edges between different LLM-invoking nodes. This formulation encapsulates both the structural components and operational parameters that define an agentic workflow’s behavior.

Given a task T and evaluation metrics L , the goal of workflow optimization is to discover the optimal workflow K^* that maximizes performance:

$$K^* = \arg \max_{K \in \mathcal{K}} L(K, T) \quad (9.3)$$

where K is the search space of workflow, and $L(K, T)$ typically measures multiple aspects including task completion quality, computational efficiency, and execution latency. This optimization objective reflects the practical challenges in deploying agentic workflows, where we must balance effectiveness with resource constraints.

9.3.2 Optimizing Workflow Edges

The edge space \mathcal{E} defines the representation formalism for agentic workflows. Current approaches primarily adopt three distinct representation paradigms: graph-based, neural network-based, and code-based structures. Each paradigm offers unique advantages and introduces specific constraints on the optimization process.

Graph-based representations enable the expression of hierarchical, sequential, and parallel relationships between nodes. This approach naturally accommodates complex branching patterns and facilitates visualization of workflow topology, making it particularly suitable for scenarios requiring explicit structural manipulation. For example, GPTSwarm [651] demonstrated the effectiveness of graph-based workflow representation in coordinating multiple LLM components through topology-aware optimization. Neural network architectures provide another powerful representation paradigm that excels in capturing non-linear relationships between nodes. Dylan [725] showed that neural network-based workflows can exhibit adaptive behavior through learnable parameters, making them especially effective for scenarios requiring dynamic adjustment based on input and feedback. Code-based representation offers the most comprehensive expressiveness among current approaches. AFLOW [773] and ADAS [741] established that representing workflows as executable code supports linear sequences, conditional logic, loops, and the integration of both graph and network structures. This approach provides precise control over workflow execution and leverages LLMs’ inherent code generation capabilities.

The choice of edge space representation significantly influences both the search space dimensionality and the applicable optimization algorithms. [728] focused solely on prompt optimization while maintaining a fixed workflow topology, enabling the use of textual feedback-based optimization techniques. In contrast, [651] developed reinforcement learning algorithms for joint optimization of individual node prompts and overall topology. [773] leveraged code-based representation to enable direct workflow optimization by language models, while recent advances by [787] and [788] introduced methods for problem-specific topology optimization.

9.3.3 Optimizing Workflow Nodes

The node space \mathcal{N} consists of four key dimensions that influence node behavior and performance. The output format space F significantly impacts performance by structuring LLM outputs, with formats like XML and JSON enabling more precise control over response structure. The temperature parameter τ controls output randomness, affecting the stability-creativity tradeoff in node responses. The prompt space P inherits the optimization domain from prompt-level optimization, determining the core interaction patterns with LLMs. The model space M represents available LLMs, each with distinct capabilities and computational costs.

For single-node optimization, existing research has primarily focused on specific dimensions within this space. [773] concentrated exclusively on prompt optimization, while [741] extended the search space to include both prompts and temperature parameters. Taking a different approach, [789] fixed prompts while exploring model selection across different nodes. Output format optimization, though crucial, remains relatively unexplored [790].

Compared to edge space optimization, node space optimization poses unique scalability challenges due to the typically large number of nodes in agentic workflows. The dimensionality of the search space grows multiplicatively with each additional node, necessitating efficient optimization strategies that can effectively handle this complexity while maintaining reasonable computational costs.

9.4 Tool Optimization

Unlike conventional usage of LLMs that typically operate in a single-turn manner, agents are equipped with advanced multi-turn planning capabilities and the ability to interact with the external world via various tools. These unique attributes make the optimization of tool usage a critical component in enhancing an agent’s overall performance and adaptability. Tool optimization involves systematically evaluating and refining how an agent selects, invokes, and integrates available tools to solve problems with higher efficiency and lower latency. Key performance metrics in this context include decision-making accuracy, retrieval efficiency, selection precision, task planning, and risk management. Central to this optimization are two complementary strategies: *tool learning* and *tool creation*.

9.4.1 Learning to Use Tools

Unlike prompting-based methods that leverage frozen foundation models’ in-context learning abilities, training-based methods optimize the model that backs LLM agents with supervision. Drawing inspiration from developmental psychology, tool learning can be categorized into two primary streams: *learning from demonstrations* and *learning from feedback* [714]. The other way to elicit the power of LLMs (agents) using tools is by using prompt-based or in-context learning methods for better reasoning abilities.

Learning from demonstrations involves training models backed LLM agents to mimic expert behaviors through imitation learning. Techniques such as behavior cloning allow models to learn policies in a supervised manner by replicating human-annotated tool-use actions. Formally, given a dataset $D = \{(q_i, a_i^*)\}_{i=0}^{N-1}$, where q_i is a user query and a_i^* is the corresponding human demonstration, the controller’s parameters θ_C are optimized as:

$$\theta_C^* = \arg \max_{\theta_C} \mathbb{E}_{(q_i, a_i^*) \in D} \prod_{t=0}^{T_i} p_{\theta_C}(a_{i,t}^* \mid x_{i,t}, H_{i,t}, q_i)$$

where $a_{i,t}^*$ is the human annotation at timestep t for query q_i , and T_i is the total number of timesteps.

Learning from feedback leverages reinforcement learning to enable models to adapt based on rewards derived from environment or human feedback. The optimization objective for the controller’s parameters θ_C is:

$$\theta_C^* = \arg \max_{\theta_C} \mathbb{E}_{q_i \in Q} \mathbb{E}_{\{a_{i,t}\}_{t=0}^{T_i}} \left[R \left(\{a_{i,t}\}_{t=0}^{T_i} \right) \right]$$

where R represents the reward function based on the sequence of actions $\{a_{i,t}\}$.

Integrating tool learning into the optimization framework enhances the system’s ability to generalize tool usage across diverse tasks and environments. By incorporating both demonstration-based and feedback-based learning, the model can iteratively improve its tool invocation strategies, selection policies, and execution accuracy.

Optimization Reasoning Strategies for Tool Using Optimizing the aforementioned metrics for better LLM agents’ abilities requires a combination of advanced retrieval models, fine-tuned reasoning strategies, and adaptive learning mechanisms. Reasoning strategies, such as Chain-of-Thought (CoT) [46], Tree-of-Thought [72], and Depth-First Search Decision Trees (DFS-DT) [690], facilitate more sophisticated decision-making processes regarding tool usage. Fine-tuning the model’s understanding of tools, including parameter interpretation and action execution, enables more precise and effective tool interactions. Additionally, learning from the model’s outputs allows for better post-processing and analysis, further refining tool utilization efficacy.

9.4.2 Creation of New Tools

Beyond the optimization of existing tools, the ability to create new tools dynamically [703, 702, 772] based on a deep understanding of tasks and current tool usage can significantly enhance the LLM Agent framework’s adaptability and efficiency. In recent work, several complementary approaches have been proposed. ToolMakers [702] establishes a closed-loop framework where a tool-making agent iteratively executes three phases: (1) *Proposing* Python functions via programming-by-example using three demonstrations, (2) *Verifying* functionality through automated unit testing (3 validation samples) with self-debugging of test cases, and (3) *Wrapping* validated tools with usage demonstrations for downstream tasks. This rigorous process ensures reliability while maintaining full automation. CREATOR [703] adopts a four-stage lifecycle: *Creation* of task-specific tools through abstract reasoning, *Decision* planning for tool invocation, *Execution* of generated programs, and *Rectification* through iterative tool refinement—emphasizing tool diversity, separation of abstract/concrete reasoning, and error recovery mechanisms. In contrast, CRAFT [772] employs an offline paradigm that distills domain-specific data into reusable, atomic tools (e.g., object color detection) through GPT-4 prompting, validation, and deduplication. Its training-free approach combines human-inspectable code snippets

with compositional problem-solving, enabling explainable toolchains while avoiding model fine-tuning—particularly effective when decomposing complex tasks into modular steps.

The integration of these complementary approaches presents rich research opportunities. Hybrid systems could merge CRAFT’s pre-made tool repositories with ToolMakers’ on-demand generation, using functional caching to balance efficiency and adaptability. Future frameworks might implement multi-tier tool hierarchies where primitive operations from CRAFT feed into ToolMakers’ composite tools, while CREATOR-style rectification handles edge cases. Advances in self-supervised tool evaluation metrics and cross-domain generalization could further automate the tool lifecycle. Notably, the interplay between tool granularity (atomic vs. composite) and reusability patterns warrants systematic investigation—fine-grained tools enable flexible composition but increase orchestration complexity. As agents evolve, bidirectional tool-task co-adaptation mechanisms may emerge, where tools reshape task representations while novel tasks drive tool innovation, ultimately enabling self-improving AI systems.

9.4.3 Evaluation of Tool Effectiveness

The evaluation metrics and benchmarks discussed below offer a comprehensive basis for quantifying an agent’s tool usage capabilities. By assessing aspects such as tool invocation, selection accuracy, retrieval efficiency, and planning for complex tasks, these benchmarks not only measure current performance but also provide clear, concrete objectives for optimizing tool usage. Such metrics are instrumental in guiding both immediate performance enhancements and long-term strategic improvements in agent-based systems. In the following sections, we first review the evolution of agent tool use benchmarks and then consolidate the key evaluation metrics that serve as targets for further tool optimization.

Tool Evaluation Benchmarks Recent efforts in LLM-as-Agent research have spawned diverse benchmarks and frameworks for evaluating tool-use capabilities. Early studies such as Gorilla [727] and API-Bank [791] pioneered large-scale datasets and methods for testing LLM interactions with external APIs, shedding light on issues like argument accuracy and hallucination. Subsequent works like T-Bench [792] and ToolBench [690] introduced more extensive task suites and stressed the importance of systematic data generation for tool manipulation. StableToolBench [793] further extended this line of inquiry by highlighting the instability of real-world APIs, proposing a virtual API server for more consistent evaluation. Meanwhile, ToolAlpaca [794] investigated the feasibility of achieving generalized tool-use in relatively smaller language models with minimal in-domain training. Additional efforts like ToolEmu [795] assessed the safety and risk aspects of tool-augmented LM agents through emulated sandbox environments. MetaTool [796] then introduced a new benchmark focused on whether LLMs know *when* to use tools and can correctly *choose* which tools to employ. It provides a dataset named ToolE that covers single-tool and multi-tool usage scenarios, encouraging research into tool usage awareness and nuanced tool selection. ToolEyes [797] pushed the evaluation further by examining real-world scenarios and multi-step reasoning across a large tool library. Finally, τ -bench [798] introduced a human-in-the-loop perspective, emphasizing dynamic user interactions and policy compliance in agent-based conversations. Together, these benchmarks and frameworks underscore the evolving landscape of tool-augmented LLM research, marking a shift from isolated reasoning tasks to comprehensive, real-world agent evaluations.

Metrics for Tool Invocation Deciding whether to invoke an external tool is a critical step that can significantly affect both the efficiency and the effectiveness of a system. In many scenarios, the model must determine if its own reasoning is sufficient to answer a query or if additional external knowledge (or functionality) provided by a tool is required. To formalize this process, we introduce a labeled dataset

$$D_{\text{inv}} = \{(q_i, y_i)\}_{i=0}^{N-1},$$

where q_i represents the i -th user query and $y_i \in \{0, 1\}$ is a binary label indicating whether tool invocation is necessary ($y_i = 1$) or not ($y_i = 0$). Based on this dataset, the model learns a decision function $d(q_i)$ defined as:

$$d(q_i) = \begin{cases} 1, & \text{if } P_\theta(y = 1 \mid q_i) \geq \tau, \\ 0, & \text{otherwise,} \end{cases}$$

where $P_\theta(y = 1 \mid q_i)$ denotes the predicted probability (from a model parameterized by θ) that a tool should be invoked for query q_i , and τ is a predetermined threshold.

In addition to this decision rule, several metrics can be used to evaluate the model’s ability to correctly decide on tool invocation. For example, the overall invocation accuracy A_{inv} can be computed as:

$$A_{\text{inv}} = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{1}\{d(q_i) = y_i\},$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. Other metrics such as precision, recall, and F1 score are also applicable. Moreover, if C_{inv} represents the cost incurred by invoking a tool and $R(q_i)$ the benefit or reward obtained when a tool is correctly used, one can define a net benefit score:

$$B_{\text{inv}} = \sum_{i=0}^{N-1} (\mathbf{1}\{d(q_i) = 1\} \cdot R(q_i) - C_{\text{inv}}).$$

This formulation not only emphasizes accuracy but also considers the cost-effectiveness of invoking external tools.

Tool Selection Among Candidates Once the decision to invoke a tool is made, the next challenge is to select the most appropriate tool from a pool of candidates. Let the candidate toolset be represented as:

$$\mathcal{T} = \{t_1, t_2, \dots, t_M\}.$$

For a given query q_i , assume that the optimal tool (according to ground truth) is t_i^* and the model selects \hat{t}_i . The simplest measure of selection performance is the tool selection accuracy A_S :

$$A_S = \frac{1}{|Q|} \sum_{q_i \in Q} \mathbf{1}\{\hat{t}_i = t_i^*\}.$$

However, many scenarios involve ranking multiple candidate tools by their relevance. In such cases, ranking-based metrics such as Mean Reciprocal Rank (MRR) and normalized Discounted Cumulative Gain (nDCG) offer a more nuanced evaluation. [690] use those two when evaluating the tool retriever system.

Tool Retrieval Efficiency and Hierarchical Accuracy Tool retrieval involves both the speed of identifying a suitable tool and the accuracy of that selection. Efficient retrieval methods reduce latency and computational overhead, while high retrieval accuracy ensures that the most relevant tool is identified for the task. To evaluate tool usage comprehensively, we adopt a hierarchical framework that distinguishes between retrieval accuracy and selection accuracy. Retrieval accuracy (A_R) reflects how precisely the system retrieves the correct tool from the repository, typically measured by metrics such as Exact Match (EM) and F1 score, which capture both complete and partial matches. In contrast, selection accuracy (A_S) assesses the system’s ability to choose the optimal tool from a set of candidates, again using similar metrics. Overall tool usage awareness is further evaluated by accuracy, recall, precision, and F1 score.

The overall retrieval efficiency E_{Ret} is thus can be expressed as:

$$E_{\text{Ret}} = \frac{A_R \times A_S \times A_P \times A_U}{C_R}$$

where C_R is the cost associated with retrieval. Optimization strategies may involve training embedding models with feedback mechanisms to enhance both efficiency and each hierarchical component of accuracy.

For a more nuanced evaluation of tool selection, Metatool [796] introduces the Correct Selection Rate (CSR), which quantifies the percentage of queries for which the model selects the expected tool(s). This evaluation framework addresses four aspects: selecting the correct tool among similar candidates, choosing appropriate tools in context-specific scenarios, ensuring reliability by avoiding the selection of incorrect or non-existent tools, and handling multi-tool queries. Together, these metrics and sub-tasks provide a robust measure of both the efficiency and precision in tool retrieval and selection.

Tool Planning for Complex Tasks Complex tasks often require the sequential application of multiple tools to reach an optimal solution. A tool plan can be represented as an ordered sequence

$$\Pi = [t_1, t_2, \dots, t_K],$$

where K is the number of steps. The quality of such a plan is typically evaluated by balancing its task effectiveness (e.g., via a metric $R_{\text{task}}(\Pi)$) against the plan’s complexity (or length). This balance can be captured by a composite planning score of the form

$$S_{\text{plan}} = \alpha \cdot R_{\text{task}}(\Pi) - \beta \cdot K,$$

where α and β are coefficients that adjust the trade-off between the benefits of high task performance and the cost associated with plan complexity. When ground truth plans Π^* are available, similarity metrics such as BLEU or ROUGE can be used to compare the predicted plan Π with Π^* , and an overall planning efficiency metric can be defined accordingly.

In addition, recent work such as ToolEyes [797] highlights the importance of behavioral planning in tool usage. Beyond selecting tools and parameters, it is crucial for LLMs to concisely summarize acquired information and strategically plan

subsequent steps. In this context, the behavioral planning capability is evaluated along two dimensions. First, the score $S_{b\text{-}validity} \in [0, 1]$ is computed by assessing (1) the reasonableness of summarizing the current state, (2) the timeliness of planning for the next sequence of actions, and (3) the diversity of planning. Second, the score $S_{b\text{-}integrity} \in [0, 1]$ is calculated by evaluating (1) grammatical soundness, (2) logical consistency, and (3) the ability to correct thinking. The composite behavioral planning score is then determined as

$$S_{BP} = S_{b\text{-}validity} \cdot S_{b\text{-}integrity},$$

providing a holistic measure of the model’s planning capability. This integrated framework ensures that tool planning for complex tasks not only focuses on the selection and ordering of tools but also on maintaining coherent, effective, and strategically sound planning processes.

In summary, optimizing tool performance within an Agent system necessitates a comprehensive approach that balances decision-making accuracy, retrieval efficiency, hierarchical selection precision, strategic planning, rigorous risk management, and robust tool learning mechanisms. By implementing targeted optimization and learning strategies, it is possible to enhance both the effectiveness and efficiency of tool-assisted machine learning workflows.

9.5 Towards Autonomous Agent Optimization

In addition to optimizing individual modules in agent evolution, such as prompts, tools, and workflows—which are susceptible to local optima that can compromise the overall performance of the agentic system, a significant body of research focuses on optimizing multiple components within the entire agentic systems. This holistic approach enables large language model (LLM) agents to evolve more comprehensively. However, optimizing the entire system imposes higher requirements. The algorithm must not only account for the impact of individual components on the agentic system but also consider the complex interactions between different components.

ADAS [741] is one of the most representative works that first formally defines the research problem of automated design in agentic systems. It integrates multiple agentic system components into the evolutionary pipeline. Specifically, ADAS introduces a meta-agent capable of iteratively designing the agentic system’s workflow, prompts, and potential tools within the overall optimization process. As demonstrated in the experiments, the automatically designed agentic systems outperform state-of-the-art hand-designed baselines.

Additionally, [726] proposes an agent symbolic learning framework for training language agents, inspired by connectionist learning principles used in neural networks. By drawing an analogy between agent pipelines and computational graphs, the framework introduces a language-based approach to backpropagation and weight updates. It defines a prompt-based loss function, propagates language loss through agent trajectories, and updates symbolic components accordingly. This method enables structured optimization of agentic workflows and naturally extends to multi-agent systems by treating nodes as independent agents or allowing multiple agents to act within a single node.

[799] proposes an approach to optimize both prompts and the agent’s own code, enabling self-improvement. This aligns with the concept of self-reference, where a system can analyze and modify its own structure to enhance performance.

Similarly, [773], [787], [800] and [788] focus on optimizing both the workflow and prompts within agentic systems. In particular, [285] introduces an approach that trains additional large language models (LLMs) to generate prompts and workflows, enabling the automated design of agentic system architectures.

In summary, optimizing the workflow of an entire agentic system is not merely a straightforward aggregation of individual component optimizations. Instead, it requires carefully designed algorithms that account for complex interdependencies among components. This makes system-wide optimization a significantly more challenging task, necessitating advanced techniques to achieve effective and comprehensive improvements.

Chapter 10

Large Language Models as Optimizers

In this chapter, we present and discuss existing works that conceptualize LLMs as optimizers. First, we note that most existing studies focus on the prompt optimization problem defined in Equation (9.1), as optimizing other components of agentic workflows remains an emerging research area. To proceed, we draw parallels with classical iterative algorithms and examine their integration into modern optimization workflows.

10.1 Optimization Paradigms

Traditional optimization methods differ in their assumptions about objective function accessibility. We categorize them into three broad classes, each with an expanding level of input space: *gradient-based optimization*, which relies on explicit function gradients; *zeroth-order optimization*, which operates without gradient information; and *LLM-based optimization*, which extends beyond numerical functions to optimize over structured and high-dimensional input spaces.

- **Gradient-Based Optimization.** These methods assume access to gradient information and iteratively refine parameters. Techniques such as stochastic gradient descent (SGD) and Newton’s method [801] are widely used but require differentiability, limiting their applicability to discrete problems like prompt tuning and structured decision workflows, often endowed with a graph structure.
- **Zeroth-Order Optimization.** These methods bypass the need for explicit gradients by estimating search directions from function evaluations [802]. Examples include Bayesian optimization [803], evolutionary strategies [804], and finite-difference methods [805], which are effective when gradients are unavailable or expensive to compute. However, they still rely on well-defined numerical objectives and structured search spaces, which constrains their applicability to language-based tasks.
- **LLM-Based Optimization.** LLMs optimize broader solution spaces by leveraging natural language as both the optimization domain and feedback mechanism. By incorporating structured reasoning and human-like iteration, LLMs excel in refining prompts, generating adaptive workflows, and iteratively improving task performance based on user feedback.

While gradient-based and zeroth-order methods are typically applied to numerical objectives, their core principles, such as iterative refinement, search heuristics, and adaptive learning, also underlie LLM-based optimization strategies. Building on these insights, we highlight a rapidly emerging class of LLM-based optimization powered by reinforcement learning, which has become the backbone of slow thinking reasoning models [90, 806, 89]. As these models continue to evolve, we anticipate them driving the next wave of agentic applications, enabling LLMs to navigate complex environments with greater adaptability and strategic foresight.

10.2 Iterative Approaches to LLM Optimization

Some LLM-based optimization methods directly draw inspiration from classical optimization theory by adapting key components to address discrete and structured challenges. A central characteristic of these approaches is the iterative update step, in which model-generated modifications are selected from a range of possible improvements to refine the objective. Using the prompt optimization objective from Equation (9.1) as a running example, a general iterative

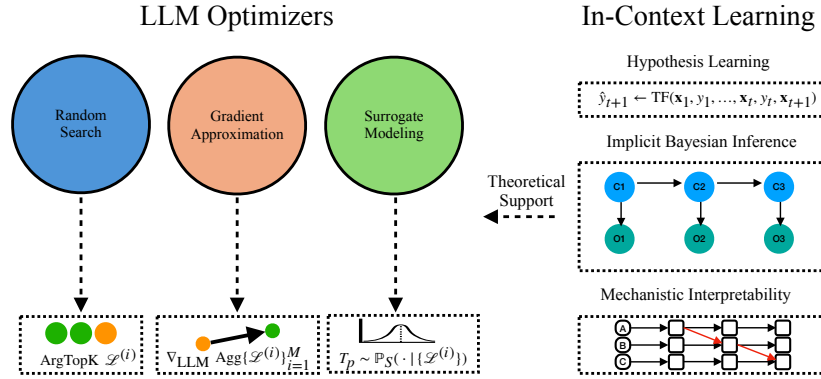


Figure 10.1: A taxonomy of LLM-based optimization methods, categorized into random search, gradient approximation, and surrogate modeling. We also highlight some theoretical explanations of in-context learning, which includes hypothesis learning, implicit Bayesian inference, and mechanistic interpretability, which underpin the optimization capabilities of LLMs.

algorithm can be expressed as follows:

Sample: $T \sim \mathcal{D}$

Evaluation: $\mathcal{L}(T; T_p) \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p), T)$

Update: $T'_p \leftarrow \phi_{\text{opt}}(\mathcal{L}(T; T_p))$

Here, the **Sample** and **Update** steps are defined based on the agent’s task. In the simplest case, such as optimizing an instruction for binary classification of movie reviews, the objective \mathcal{L} is measured by classification accuracy. In more complex agentic workflows, the decision variable may include prompts at different workflow stages, tool selections, agent topologies, or a combination thereof. As discussed in Chapter 9, a common characteristic of these decision variables is their *combinatorial* nature—such as the set of all strings from an LLM’s vocabulary \mathcal{V} or all possible role assignments for agents in a workflow. Since enumerating all possible solutions is often intractable, this necessitates designing approximate update steps ϕ_{opt} , which we discuss next.

- **Random Search.** Early LLM-based optimization methods leveraged random search variants to optimize prompts in discrete natural language spaces [774, 807, 651, 732, 808, 809, 810]. These methods often resemble evolutionary algorithms that iteratively sample candidate decision variables and select the top-performing ones from each iteration. The general formulation follows:

Sample: $T \sim \mathcal{D}$

Evaluation: $\mathcal{L}^{(i)} \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p^{(i)}), T), \quad i = 1, \dots, M$

Update: $\{T_p^{(k)}\}_{k=1}^K \leftarrow \text{ArgTopK}_{i \in [M]} \mathcal{L}^{(i)},$

Replenishment (Optional): $\{T_p^{(j)}\}_{j=K+1}^M \sim \text{Mutate}(\{T_p^{(k)}\}_{k=1}^K).$

We briefly override previous notations and let M denote the total number of candidate prompts sampled per iteration, and K (with $K < M$) control the number of top-performing candidates—selected with ArgTopK in our algorithm—retained for the next step. This algorithm can optionally incorporate a replenishment step to maintain diversity in the candidate pool across iterations. Random search methods are simple to implement, highly parallelizable, and particularly effective for single-prompt workflows. Beyond prompt optimization, they have also demonstrated strong performance in selecting in-context demonstrations [811, 812]. However, their efficiency comes at a cost—each iteration requires $O(M)$ parallel API queries, which can become prohibitively expensive for complex workflows involving multiple queries.

- **Gradient Approximations.** Several methods approximate gradient-based updates by iteratively refining solutions. For instance, [779, 730, 728] generate refinements at different workflow stages. StraGO [722] estimates descent directions using central-difference heuristics, while Trace [813] optimizes composed programs by modeling them as computation graphs, similar to backpropagation. The key analogy between gradient updates in continuous optimization and prompt-space refinement is the concept of a “descent direction”—a systematic *modification* of the decision variable to improve the objective. In contrast, random search methods propose new decision variables independently at each step, without accessing past update trajectories. Gradient-based approaches, by contrast, exploit this historical information, often leading to faster convergence. A general iteration for gradient approximation methods is given below:

$$\begin{aligned}
&\textbf{Sample: } T^{(i)} \sim \mathcal{D}, \quad i = 1, \dots, M \\
&\textbf{Evaluation: } \mathcal{L}^{(i)} \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p), T^{(i)}), \quad i = 1, \dots, M \\
&\textbf{Gradient Approximation: } g \leftarrow \nabla_{\text{LLM}} \text{Agg}(\mathcal{L}^{(1)}, \dots, \mathcal{L}^{(M)}) \\
&\textbf{Update: } T'_p \leftarrow \phi_{\text{opt}}(T_p, g),
\end{aligned}$$

where M is the minibatch size, $\text{Agg}(\cdot)$ is an aggregation function that combines feedback signals (e.g., in numerical optimization, Agg is typically the average operator), ∇_{LLM} represents an abstract “LLM-gradient operator” [728] that generates textual refinement directions based on the feedback signal and the current minibatch (e.g., *the agent should consider the edge case of ...*). Additionally, ϕ_{opt} can be instantiated as an LLM query, allowing the agent to update its prompt based on g .

Compared to random search methods, gradient-based approaches offer two key advantages: they enable the incorporation of past refinement directions into ϕ_{opt} , analogous to momentum-based techniques in first-order optimization algorithms [814, 815], and they facilitate backpropagation-like techniques for optimizing computation graphs [651, 813, 780], making them particularly effective for multi-stage workflows with interdependent optimizable modules. However, this flexibility comes at the cost of increased design overhead, such as the need for meta-prompts to aggregate feedback and apply refinement directions. We further discuss the feasibility of using LLMs to optimize these *hyperparameters* below. Some approaches also explored direct gradient-based optimization of soft prompts [816, 817, 818]. While effective for simple input-output sequence learning, these methods struggle with multi-step workflows and sequential decision-making [630, 300].

Finally, while these methods leverage first-order optimization insights, the extension of second-order techniques (e.g., quasi-Newton methods) to LLM-based optimization remains largely unexplored. Fortunately, recent works such as Revolve [780] have taken a step in this direction by introducing a structured approach for second-order optimization, modeling the evolution of response patterns over multiple iterations. By incorporating higher-order refinements, Revolve enables more stable and informed optimization, effectively mitigating stagnation in complex tasks. We are also excited by emerging trends in leveraging inference-time compute [90, 89] to incorporate historical refinement directions and investigate the benefits of momentum.

- **Bayesian Optimization and Surrogate Modeling.** While the aforementioned approaches achieved significant progress in LLM-based optimization, they often entail substantial financial and environmental costs due to the high number of required LLM interactions. Moreover, these methods can be sensitive to noise, and the optimization landscape of discrete prompts, among other decision variables, remains poorly understood [819, 820]. Under these constraints, Bayesian Optimization (BO) emerges as a compelling alternative, as it builds a noise-resilient surrogate model of the optimization objective:

$$\begin{aligned}
&\textbf{Sample: } T \sim \mathcal{D} \\
&\textbf{Proposal: } \{T_p^{(i)}\}_{i=1}^M \sim S. \text{Propose} \\
&\textbf{Evaluation: } \mathcal{L}^{(i)} \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p^{(i)}), T), \quad i = 1, \dots, M \\
&\textbf{Update: } S \leftarrow S. \text{UpdatePrior}(\{\mathcal{L}^{(i)}\}_{i=1}^M, \{T_p^{(i)}\}_{i=1}^M),
\end{aligned}$$

where S represents a probabilistic surrogate model of the optimization objective, equipped with a proposal operator (e.g., posterior sampling from a Gaussian Process BO procedure [803]) and an update mechanism based on observed evidence from prompt evaluations. For instance, MIPRO [821] employs a Tree-Structured Parzen Estimator as its surrogate [822], while PROMST [823] trains a score-prediction model to guide prompt tuning. Leveraging a surrogate model for LLM-based optimization aligns with the emerging trend of amortized optimization for non-differentiable objectives [824]. For instance, [825] trains a prompt-generator LLM to amortize the computational cost of instantiating a beam search problem for discovering jailbreak attack prefixes.

Finally, several other works fit an additional lightweight module—such as a Bayesian belief posterior or a utility function—from LLM outputs, to aid the optimization of domain-specific workflows, such as decision-making and multi-agent negotiations [826, 827]. This type of amortized methods—those that fit a parameterized model that is reusable for unseen inputs—have found increasing usage in LLM-based optimization, such as jailbreaking [828, 825].

10.3 Optimization Hyperparameters

Similar to traditional optimization, LLM-based methods are highly sensitive to hyperparameters that influence search efficiency and generalization. A key consideration in gradient-based LLM optimizers is the choice of the aggregation function $\text{Agg}(\cdot)$, which determines how textual feedback is synthesized to guide prompt updates. An improper choice can lead to loss of critical information or misalignment in iterative refinements. Additionally, [813] introduces a “whiteboard” approach, where an LLM program is decomposed into human-interpretable modules. However, design choices in structuring such modular workflows remain largely unexplored, which poses an open challenge for optimizing LLM-driven decision-making pipelines.

Hyperparameters in LLM optimization often parallel those in numerical optimization. For example, batch size plays a crucial role: just as minibatch updates enhance stability and efficiency in classical optimization, LLM-based approaches like TextGrad [728] aggregate feedback across multiple generated samples before making updates. Another key factor is momentum—while it stabilizes updates in gradient-based methods by incorporating past gradients, LLM-based optimizers similarly leverage historical refinements to improve performance over time [728, 813]. Despite progress in numerical optimization, hyperparameter selection for LLM-based optimizers remains largely heuristic, often relying on ad hoc, trial-and-error tuning.

In agentic system design, hyperparameters proliferate across various components, including role assignments of agents, selection of in-context demonstrations, and scheduling of tool invocations. Each of these choices has a profound impact on downstream performance, yet principled methods for optimizing them remain underdeveloped. While traditional hyperparameter tuning techniques, such as grid search and Bayesian optimization, can be applied to discrete LLM-driven workflows, their computational cost scales poorly due to the high variance in language model outputs. Additionally, the combinatorial nature of these hyperparameters, where agent configurations, prompting strategies, and reasoning structures interact in complex ways, makes an exhaustive search infeasible. Recent work has attempted to address this challenge by embedding agentic workflows into structured frameworks such as finite state machines [729], optimal decision theory [826], and game theory [827]. However, these approaches often fail to generalize across diverse environments. A promising direction for addressing these challenges is meta-optimization, where LLMs are used to optimize their own hyperparameters and decision-making strategies. For example, an LLM-based optimizer can iteratively refine its own prompting strategies by treating past decisions as experience, akin to learned optimizers in deep learning [829]. Moreover, amortized approaches train auxiliary models to predict effective hyperparameters, which can reduce the computational cost of exhaustive search [821, 823]. While these techniques offer exciting possibilities, they also introduce new challenges, such as balancing exploration with exploitation in adaptive tuning and ensuring generalization across diverse optimization tasks. Investigating principled meta-optimization strategies tailored to LLM-driven workflows remains a critical area for future research.

10.4 Optimization across Depth and Time

Unlike conventional optimizers that update parameters in a static setting, LLMs optimize workflows dynamically, considering both depth (single-pass workflows) and time (recurrent updates). In terms of depth, LLMs function similarly to feedforward networks, sequentially optimizing workflows as they pass through different modules—most existing LLM-based optimizers follow this paradigm. Beyond single-pass execution, LLMs can also optimize over time, akin to recurrent architectures such as RNNs or Universal Transformers [830], by iteratively refining decision-making. For instance, StateFlow [729] enhances workflows by incorporating feedback across multiple iterations, enabling dynamic refinement and adaptation over time. While these analogies are compelling, many well-established engineering optimization techniques—such as checkpointing [831] and truncated backpropagation [832]—remain underexplored in LLM-based optimization. We see this as a promising avenue for future research, echoing previous calls for deeper investigation [813].

10.5 A Theoretical Perspective

Recent studies suggest that transformers inherently perform optimization-like computations, supporting their potential as general-purpose optimizers for computational workflows. However, a significant gap remains between their empirical success and theoretical understanding. Here, we provide a brief overview of recent progress in bridging this gap.

- **In-Context Learning.** A fundamental perspective on transformers as optimizers emerges from in-context learning, particularly in few-shot settings [2]. [733] demonstrated that transformers can in-context learn diverse regression hypotheses, including regularized linear models, decision trees, and shallow neural networks. Building on this, later works [734, 833, 735] provided constructive proofs that transformers can implement iterative optimization algorithms, such as gradient descent and second-order updates. However, while these theoretical models characterize transformers’ optimization capabilities, they do not fully explain in-context learning in large-scale LLMs, which operate in discrete input-output spaces. Empirical analyses [819, 834, 820] instead sought to understand how pre-trained LLMs generalize in-context. [834] proposed that in-context learning resembles a hidden Markov model (HMM) performing implicit Bayesian inference, while [819, 820] challenged the conventional view that in-context demonstrations serve as new test-time samples for hypothesis formation. In-context learning remains the central emergent ability [835] enabling self-improvement and optimization from context, yet it continues to elude comprehensive theoretical analysis.
- **Mechanistic Interpretability.** Parallel to theoretical analyses, mechanistic interpretability aims to uncover internal transformer computations by identifying subgraphs, also known as circuits, responsible for specific behaviors. Early studies mapped circuits for stylized language tasks in pre-trained GPT-2 models [836, 837, 838], while more recent efforts have scaled up by identifying semantically meaningful features using sparse autoencoders [839, 736, 840, 841]. These methods have been largely successful in eliciting causal and controllable behavior from frontier-class LLMs, but they also reveal an unintended consequence: in-context learning capabilities often entangle beneficial generalization with harmful behaviors when conditioned on many-shot demonstrations [842]. This raises challenges for optimizing LLM workflows safely and reliably.
- **Limitations Under Uncertainty.** While LLMs demonstrate moderate capabilities in sequential decision-making when provided with in-context information, they struggle to make optimal choices under uncertainty [843, 844, 845, 846]. In particular, [826] found that LLM-based optimizers exhibit difficulty in adapting to stochastic environments, often failing to explore optimally. These findings serve as a cautionary note for deploying LLM-based optimizers in dynamic or uncertain settings where exploration and robust decision-making are critical.

LLMs redefine optimization by integrating structured reasoning, natural language processing, and in-context learning, expanding beyond traditional numerical methods. Despite strong empirical performance in structured search spaces, open questions remain about the theoretical underpinnings of LLM-based optimization, particularly the emergence of in-context learning from standard gradient-based training.

Chapter 11

Online and Offline Agent Self-Improvement

In the pursuit of self-improvement, intelligent agents leverage optimization as both a mechanism for refining individual components—such as prompt design, workflow orchestration, tool utilization, reward function adaptation, and even the optimization algorithms themselves—and as a strategic framework that ensures these individual improvements are aligned toward coherent performance enhancement. For instance, optimizing the reward function and prompt design in isolation might yield conflicting outcomes, but a strategic approach coordinates these optimizations to maintain coherence and maximize overall effectiveness. We categorize self-evolution into two primary paradigms: *online* and *offline* self-improvement. Additionally, we explore *hybrid* optimization strategies that integrate both approaches to maximize efficiency and adaptability.

11.1 Online Agent Self-Improvement

Online self-improvement refers to real-time optimization in which an agent dynamically adjusts its behavior based on immediate feedback. This paradigm ensures that agents remain responsive to evolving environments by continuously optimizing key performance metrics—such as task success, latency, cost, and stability—in an iterative feedback loop. Online self-improvement is particularly effective in applications that require dynamic adaptability, such as real-time decision-making, personalized user interactions, and automated reasoning systems. Key optimization strategies in online self-improvement can be classified into the following four categories: Iterative Feedback and Self-Reflection, Active Exploration in Multi-Agent Systems, Real-Time Reward Shaping, and Dynamic Parameter Tuning.

Iterative Feedback and Self-Reflection These methodologies [48, 67, 72, 70, 847, 47] focus on enabling agents to critique and refine their own outputs iteratively. Reflexion [48], Self-Refine [67], and Tree of Thoughts [72] introduce self-critique loops, where the model identifies errors and proposes revisions in real-time. ReAct [70] combines chain-of-thought “reasoning” with “acting”, allowing the model to revise steps iteratively after observing external feedback. In addition, other methods either rely on self-consistency [78] to select the most coherent solution or leverage a process reward model (PRM) Lightman et al. [847] to choose the best solution from the candidates. Collectively, these frameworks reduce error propagation and support rapid adaptation without requiring a separate offline fine-tuning cycle.

Active Exploration in Multi-Agent Systems These approaches [626, 848, 627, 152] actively explore and dynamically search for novel patterns and workflow improvements in multi-agent systems. MetaGPT [626], CAMEL [848], and ChatDev [627] showcase multi-role or multi-agent ecosystems that interact in real-time, exchanging continuous feedback to refine each other’s contributions. Similarly, HuggingGPT [152] coordinates specialized models (hosted on Hugging Face) through a central LLM controller, which dynamically routes tasks and gathers feedback. These collaborative strategies further highlight how online updates among agents can incrementally refine collective outcomes.

Real-Time Reward Shaping Rather than relying on fixed or purely offline reward specifications, some frameworks [731, 91, 105, 849] integrate immediate feedback signals not only to correct errors, but also to adapt internal reward functions and policies. This enables self-adaptive reward calibration that balances trade-offs between performance, computational cost, and latency, allowing agents to optimize reward mechanisms dynamically in response to user interactions.

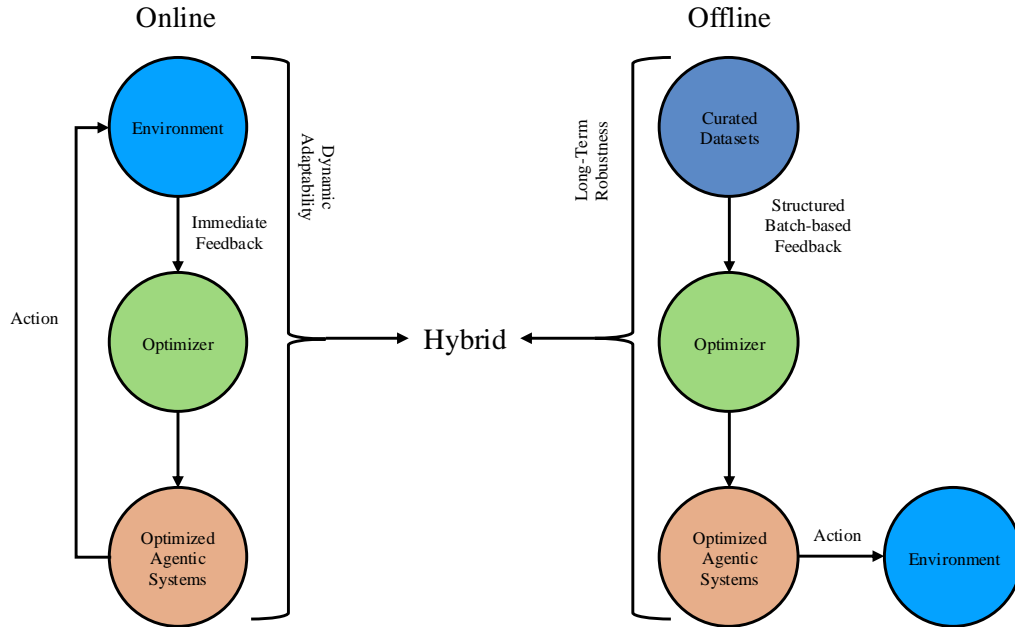


Figure 11.1: An illustration of self-improvement under three different utilization scenarios, including Online, Offline, and Hybrid self-improvement.

Dynamic Parameter Tuning In this category, agents autonomously update their internal parameters (including prompt templates, tool invocation thresholds, search heuristics, etc.) in real time, leveraging gradient-free or approximated gradient methods. These updates optimize both computational efficiency and decision accuracy, allowing for seamless adaptation to evolving contexts. Self-Steering Optimization (SSO) [850] eliminates the need for manual annotation and maintains signal accuracy while keeping training on-policy by autonomously generating preference signals during iterative training.

Online self-improvement fosters a continuously evolving agent framework where learning is embedded within task execution, promoting enhanced real-time adaptability, user-centric optimization, and robust problem-solving capabilities.

11.2 Offline Agent Self-Improvement

Offline self-improvement, in contrast, leverages structured, batch-based optimization. This paradigm utilizes scheduled training sessions with high-quality curated datasets to systematically improve the agent’s generalization capabilities [851, 667, 852, 853, 854]. Unlike online approaches, offline approaches accommodate more computationally intensive methodologies, including Batch Parameter Updates and Fine-Tuning, Meta-Optimization, and Systematic Reward Model Calibration.

Batch Parameter Updates and Fine-Tuning In this category, agents undergo extensive fine-tuning using supervised learning or reinforcement learning (RL) techniques, optimizing performance across large-scale datasets over multiple training epochs. Retrieval-augmented generation (RAG) is often integrated to enhance contextual understanding and long-term memory retrieval [740, 741]. Such methods allow agents to optimize retrieval strategies, thereby improving reasoning over extensive knowledge corpora.

Meta-Optimization of Agent Components Here offline training is not limited to improving task performance but extends to refining optimization algorithms themselves. Meta-learning strategies that optimize hyperparameters or even restructure the optimization process dynamically have demonstrated promising outcomes [731, 91]. These meta-optimization approaches enable agents to discover the most effective learning parameters for new problem domains.

Systematic Reward Model Calibration Offline settings facilitate the precise calibration of reward models, incorporating hierarchical or listwise reward integration frameworks (e.g., LIRE [855]) to align agent behavior with long-term objectives through gradient-based reward optimization. Such calibration ensures that reward functions reflect real-world task complexity, thereby mitigating bias and enhancing generalization.

The structured nature of offline optimization results in a robust agent baseline, whose performance is fine-tuned to optimize stability, efficiency, and computational cost before real-world deployment. Offline training allows for high-fidelity model refinement and is essential for mission-critical applications requiring predictable performance guarantees.

11.3 Comparison of Online and Offline Improvement

Online and offline optimization offer complementary benefits, each excelling in different aspects of self-improvement. Online optimization thrives in dynamic environments, where real-time feedback enables continuous adaptation. It is well-suited for applications that require immediate responsiveness, such as interactive agents, real-time decision-making, and reinforcement learning systems. However, frequent updates may introduce instability or drift, requiring mechanisms to mitigate performance degradation over time.

In contrast, offline optimization emphasizes structured, high-fidelity training using pre-collected datasets, ensuring robust and stable performance before deployment. By leveraging computationally intensive learning methods such as batch training, fine-tuning, and meta-optimization, offline approaches provide strong generalization and long-term consistency. However, they lack the agility of online learning and may struggle to adapt efficiently to novel scenarios without additional retraining. Table 11.1 summarizes the key distinctions between these two paradigms.

Feature	Online Optimization	Offline Optimization
Learning Process	Continuous updates based on real-time feedback	Batch updates during scheduled training phases
Adaptability	High, capable of adjusting dynamically	Lower, adapts only after retraining
Computational Efficiency	More efficient for incremental updates	More resource-intensive due to batch training
Data Dependency	Requires real-time data streams	Relies on curated, high-quality datasets
Risk of Overfitting	Lower due to continuous learning	Higher if training data is not diverse
Stability	Potentially less stable due to frequent updates	More stable with controlled training settings

Table 11.1: Comparison of Online vs. Offline Optimization Strategies in Self-Improvement Agents.

While both approaches have inherent strengths and trade-offs, modern intelligent systems increasingly integrate them through hybrid optimization strategies. These hybrid frameworks leverage the stability of offline training while incorporating real-time adaptability, enabling agents to maintain long-term robustness while continuously refining their performance in dynamic environments.

11.4 Hybrid Approaches

Recognizing that both online and offline methods have inherent limitations, many contemporary systems adopt *hybrid* optimization strategies. These hybrid methods integrate structured offline optimization with responsive online updates to achieve continuous incremental agent enhancement.

Hybrid optimization explicitly supports self-improvement by empowering agents to autonomously evaluate, adapt, and enhance their behaviors through distinct yet interconnected stages:

- **Offline Pre-Training:** In this foundational stage, agents acquire robust baseline capabilities through extensive offline training on curated datasets. This stage establishes essential skills, such as reasoning and decision-making, required for initial autonomous performance. For instance, frameworks such as the one introduced by Schrittwieser et al. [856] illustrate how offline pretraining systematically enhances initial agent capabilities, ensuring subsequent online improvements are built upon a stable foundation.
- **Online Fine-Tuning for Dynamic Adaptation:** Agents actively refine their capabilities by autonomously evaluating their performance, identifying shortcomings, and dynamically adjusting strategies based on real-time feedback. This adaptive fine-tuning stage directly aligns with the agent self-improvement paradigm by allowing real-time

optimization of agent-specific workflows and behaviors, exemplified by Decision Mamba-Hybrid (DM-H) [857], where agents efficiently adapt to complex, evolving scenarios.

- **Periodic Offline Consolidation for Long-Term Improvement:** periodic offline consolidation phases, agents systematically integrate and solidify improvements identified during online interactions. This ensures that incremental, online-acquired skills and improvements are systematically integrated into the agent’s core models, maintaining long-term stability and effectiveness. The Uni-O4 framework [858] exemplifies how this process enables seamless transitions between offline knowledge consolidation and online adaptive improvements.

Hybrid optimization thus explicitly supports autonomous, continuous evolution by seamlessly interweaving structured offline learning with proactive, real-time online adaptation. This cyclical approach equips agents with both immediate responsiveness and stable long-term improvement, making it ideally suited for complex, real-world scenarios such as autonomous robotics, personalized intelligent assistants, and interactive systems.

Chapter 12

Scientific Discovery and Intelligent Evolution

In previous chapters, we primarily discussed the evolution of agentic systems from a technical perspective, focusing on how to develop systems that can effectively perform well-defined tasks traditionally executed by humans. However, a fundamental and important question remains: can these agents drive a self-sustaining innovation cycle that propels both agent evolution and human progress?

Scientific knowledge discovery is a compelling example of self-evolution in intelligent beings, as it helps them adapt to the world in a sustainable way. Agents capable of discovering scientific knowledge at different levels of autonomy and in a safe manner will also play important roles in technological innovation for humanity. In this section, we survey progress in autonomous discovery using agentic workflows and discuss the technological readiness toward fully autonomous, self-evolving agents. Within this scope, the goal of the agent is to uncover, validate, and integrate data, insights, and principles to advance an objective scientific understanding of natural phenomena. Instead of altering the world, the agent seeks to better understand nature as a Scientist AI [859] and assist humans in extending the boundaries of knowledge.

We first define the concept of knowledge and intelligence to clarify our discussion, then introduce three typical scenarios where agents and scientific knowledge interact. We also highlight existing successes and examples of self-enhancing agents applied to theoretical, computational, and experimental scientific research. Lastly, we summarize the current challenges for a future outlook.

12.1 Agent’s Intelligence for Scientific Knowledge Discovery

Knowledge, traditionally defined as *justified true belief*, traces back to Plato [860] and has been further refined by Edmund Gettier [861], who argued that knowledge must be produced by a reliable cognitive process—though its precise definition remains debated [862]. In our discussion, we describe scientific knowledge discovery as the process of collecting data and information to either justify or falsify rational hypotheses about target scientific problems. To discuss the capability of agents in scientific knowledge discovery, we first explore a general framework for measuring an agent’s intelligence through the lens of information theory.

12.1.1 KL Divergence-based Intelligence Measure

The agent’s intelligence can be measured by the KL divergence between its predicted and real-world probability distributions of unknown information. A long-standing goal in both artificial intelligence and the philosophy of science is to formalize what it means for an agent to “understand” the world. From Jaynes’ view of probability theory as extended logic for reasoning under uncertainty [863], to Parr et al.’s framing of intelligence as minimizing model-world divergence under the free energy principle [864], many frameworks converge on a common theme: intelligent behavior arises from making accurate predictions about an uncertain world. Clark [344], for instance, argues that intelligent agents constantly engage with the world through prediction and error correction to reduce surprise. Chollet [865] emphasizes that intelligence should reflect skill-acquisition efficiency, because of the dynamic nature of task adaptation. Together, these views suggest that intelligence involves building predictive and adaptable models—an idea formalized here through a probabilistic framework that links reasoning to knowledge acquisition and enables comparison across agents in scientific discovery.

Building on this foundation, we consider intelligence in the specific context of scientific knowledge discovery, where the agent’s primary objective is to infer unknown aspects of the physical world from limited data. From the agent’s perspective in knowledge discovery, the world \mathcal{W} is characterized by an ensemble of datasets $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ related to the scientific problem the agent aims to understand. During the agent’s interaction with \mathcal{W} , each dataset appears in the experimental measurements or observations with a probability $P_{\mathcal{W}}(\mathbf{x})$. Here we assume that individual data points x_i may or may not be correlated. For example, in a task of text generation using a language model, x_i represents a chunk of tokens forming a meaningful proposition, and \mathbf{x} is a coherent text constructed from known and inferred propositions. In this context, the “world” is the ensemble of all propositions.

Let θ denote the parameter that parameterizes the agent’s world model, M_t^{wm} , as defined in Table 1.2. For instance, in a transformer model with a fixed architecture, θ represents its weights. Given θ and a dataset \mathbf{x} , the agent predicts a probability distribution $P_{\theta}(\mathbf{x})$. In general, different AI agents could be optimized for different goals. For scientific knowledge discovery, we assume that the agent’s goal is to produce a good description of the real world, i.e., a world model that predicts yet-to-be-explored natural phenomena as accurately as possible. A more intelligent agent produces a better approximation of the real-world distribution $P_{\mathcal{W}}(\mathbf{x})$. The agent’s intelligence can thus be measured by the KL divergence, or relative entropy, between these two probability distributions:

$$D_0(\theta) = \sum_{\mathbf{x} \subseteq \mathcal{W}} P_{\mathcal{W}}(\mathbf{x}) \log \frac{P_{\mathcal{W}}(\mathbf{x})}{P_{\theta}(\mathbf{x})} \quad (12.1)$$

$D_0(\theta)$ describes the difference between $P_{\mathcal{W}}(\mathbf{x})$ and $P_{\theta}(\mathbf{x})$. More precisely, in the context of hypothesis testing, if we sample $P_{\mathcal{W}}(\mathbf{x})$ N times and compare the results with the predictions from $P_{\theta}(\mathbf{x})$, the probability of mistaking $P_{\mathcal{W}}(\mathbf{x})$ for $P_{\theta}(\mathbf{x})$ scales as $e^{-ND_0(\theta)}$ [866]. In other words, an agent with a lower $D_0(\theta)$ produces predictions that align more closely with reality.

For example, consider two materials synthesis agents whose goal, M_t^{goal} , is to understand whether or not an inorganic compound of interest, $\text{CaFe}_2(\text{PO}_4)_2\text{O}$, is synthesizable. The agents can predict either (1) $\mathbf{x}_1 = \{\text{CaFe}_2(\text{PO}_4)_2\text{O} \text{ is synthesizable}\}$, and (2) $\mathbf{x}_2 = \{\text{CaFe}_2(\text{PO}_4)_2\text{O} \text{ is not synthesizable}\}$. In reality, since $\text{CaFe}_2(\text{PO}_4)_2\text{O}$ is a natural mineral, $P_{\mathcal{W}}(\mathbf{x}_1) = 1$ and $P_{\mathcal{W}}(\mathbf{x}_2) = 0$. However, this mineral was only recently reported on October 4, 2023[ref], after the knowledge cutoff of many LLMs; thus, the agents lack that knowledge. Compare Agent 1, which guesses randomly $P_{\theta_1}(\mathbf{x}_1) = P_{\theta_1}(\mathbf{x}_2) = 0.5$, yielding $D_0(\theta_1) = \log 2$. In contrast, Agent 2 uses first-principles calculations and finds that $\text{CaFe}_2(\text{PO}_4)_2\text{O}$ (assume structure is xx [cite: Materials Project ID]) is the lowest-energy phase among its competitors [ref], indicating stability. Thereby, Agent 2 predicts that $\text{CaFe}_2(\text{PO}_4)_2\text{O}$ is likely synthesizable, suggesting $P_{\theta_2}(\mathbf{x}_1) > 0.5 > P_{\theta_2}(\mathbf{x}_2)$. Consequently, $D_0(\theta_2) = -\log P_{\theta_2}(\mathbf{x}_1) < D_0(\theta_1)$, meaning that Agent 2 has a more accurate understanding of the real world.

Now, let us assume the agent has conducted some measurements and determined specific values for a subset of data points x_i . Let \mathbf{x}_K denote this known subset and \mathbf{x}_U the remaining unknown part. Correspondingly, we define the space of all existing knowledge as \mathcal{K} and the space of all unknown information as \mathcal{U} , satisfying $\mathbf{x}_K \subseteq \mathcal{K}$, $\mathbf{x}_U \subseteq \mathcal{U}$, and $\mathcal{K} \cup \mathcal{U} = \mathcal{W}$. For example, in text generation, the the prompt text \mathbf{x}_K represents already known information. The efficiency of the language model is then measured by its predictive accuracy for the generated text \mathbf{x}_U based on \mathbf{x}_K . More generally, the agent’s intelligence is measured by the relative entropy of the conditional probability distribution:

$$D_K(\theta, \mathbf{x}_K) = \sum_{\mathbf{x} \subseteq \mathcal{U}} P_{\mathcal{W}}(\mathbf{x}|\mathbf{x}_K) \log \frac{P_{\mathcal{W}}(\mathbf{x}|\mathbf{x}_K)}{P_{\theta}(\mathbf{x}|\mathbf{x}_K)} \quad (12.2)$$

In practice, all of the agent’s knowledge is stored in its memory M_t^{mem} , i.e., $\mathbf{x}_K = \mathcal{K} = M_t^{\text{mem}}$ and $\mathcal{U} = \mathcal{W} \setminus M_t^{\text{mem}}$, we define the agent’s intelligence as:

$$IQ_t^{\text{agent}} \equiv -D_K(\theta, M_t^{\text{mem}}) = - \sum_{\mathbf{x} \subseteq \mathcal{U}} P_{\mathcal{W}}(\mathbf{x}|M_t^{\text{mem}}) \log \frac{P_{\mathcal{W}}(\mathbf{x}|M_t^{\text{mem}})}{P_{\theta}(\mathbf{x}|M_t^{\text{mem}})} \quad (12.3)$$

In other words, the the agent’s intelligence IQ_t^{agent} is determined by its memory M_t^{mem} and the parameter θ of its world model M_t^{wm} . A schematic plot is shown in Figure 12.1. At time $t = 0$, when the M_t^{mem} is very limited or lack relevant information to a new target scientific problem, IQ_t^{agent} is primarily determined by the zero-shot predictive ability of M_t^{wm} , corresponding to fluid intelligence [867]. Over time, as more relevant knowledge is incorporated into M_t^{mem} , IQ_t^{agent} becomes increasingly dependent on the knowledge-augmented predictive capability of M_t^{wm} , reflecting crystallized intelligence [868].

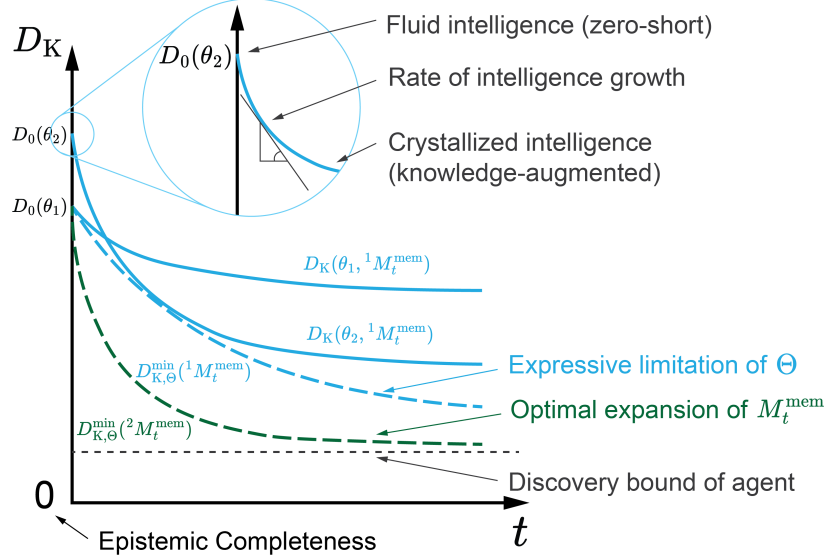


Figure 12.1: **Schematic representation of agent intelligence and knowledge discovery.** The agent’s intelligence, measured by the KL divergence D_K between predictions and real-world probability distributions, evolves from fluid intelligence (zero-shot predictions for new problems) to crystallized intelligence (knowledge-augmented predictions after learning) as it accumulates data in its memory M_t^{mem} over time t . Given M_t^{mem} , the evolution of D_K varies within the world model’s parameter space Θ , as illustrated by θ_1 and θ_2 in the solid lines. The expressive limitation of Θ is characterized by the envelope $D_{K,\Theta}^{\min}$. Given Θ , $D_{K,\Theta}^{\min}$ is influenced by different knowledge expansion strategies, such as ${}^1M_t^{\text{mem}}$ and ${}^2M_t^{\text{mem}}$, shown as dash lines.

12.1.2 Statistical Nature of Intelligence Growth

The agent’s intelligence, in a statistical sense, is a non-decreasing function of acquired knowledge. Roughly speaking, IQ_t^{agent} quantifies both the amount of knowledge an agent has acquired and how effectively the agent can apply that knowledge after learning from M_t^{mem} . Intuitively, if the agent gains additional information at time t —which corresponds to enlarging M_t^{mem} and shrinking \mathcal{U} —its intelligence should increase.

To understand this process, consider a small region $\Delta \subseteq \mathcal{U}$ and examine the effect of adding a dataset \mathbf{x}_Δ from Δ to M_t^{mem} . Denote $\mathcal{U} = \mathcal{U}' \cup \Delta$, where \mathcal{U}' represents the remaining unknown part of the world. The agent’s intelligence at time $t + 1$ is given by:

$$IQ_{t+1}^{\text{agent}} \equiv -D_K(\theta, M_t^{\text{mem}} \mathbf{x}_\Delta) = - \sum_{\mathbf{x}' \subseteq \mathcal{U}'} P_{\mathcal{W}}(\mathbf{x}' | M_t^{\text{mem}} \mathbf{x}_\Delta) \log \frac{P_{\mathcal{W}}(\mathbf{x}' | M_t^{\text{mem}} \mathbf{x}_\Delta)}{P_\theta(\mathbf{x}' | M_t^{\text{mem}} \mathbf{x}_\Delta)} \quad (12.4)$$

Directly comparing IQ_t^{agent} and IQ_{t+1}^{agent} is challenging. Instead, we can compare the expected value of IQ_{t+1}^{agent} , averaging over \mathbf{x}_Δ with probability $P_{\mathcal{W}}(\mathbf{x}_\Delta | M_t^{\text{mem}})$. This expectation represents the average amount of knowledge gained by measuring Δ , given prior knowledge in M_t^{mem} . We obtain:

$$\begin{aligned} \sum_{\mathbf{x} \subseteq \Delta} P_{\mathcal{W}}(\mathbf{x} | M_t^{\text{mem}}) IQ_{t+1}^{\text{agent}} &= - \sum_{\mathbf{x}' \subseteq \mathcal{U}', \mathbf{x} \subseteq \Delta} P_{\mathcal{W}}(\mathbf{x}' \mathbf{x} | M_t^{\text{mem}}) \log \frac{P_{\mathcal{W}}(\mathbf{x}' | M_t^{\text{mem}} \mathbf{x})}{P_\theta(\mathbf{x}' | M_t^{\text{mem}} \mathbf{x})} \\ &= IQ_t^{\text{agent}} + \sum_{\mathbf{x} \subseteq \Delta} P_{\mathcal{W}}(\mathbf{x} | M_t^{\text{mem}}) \log \frac{P_{\mathcal{W}}(\mathbf{x} | M_t^{\text{mem}})}{P_\theta(\mathbf{x} | M_t^{\text{mem}})} \end{aligned} \quad (12.5)$$

The second term is the relative entropy of the conditional probability distribution of \mathbf{x}_Δ conditioned on M_t^{mem} , which is always non-negative. Therefore, on average, IQ_{t+1}^{agent} is non-decreasing as M_t^{mem} acquires new knowledge over time. Note that IQ_{t+1}^{agent} can be further increased by leveraging the newly acquired knowledge to optimize θ within M_t^{wm} .

Interestingly, the expected gain in intelligence at time t is determined by the discrepancy between the actual distribution $P_{\mathcal{W}}(\mathbf{x} | M_t^{\text{mem}})$ and the model-predicted distribution $P_\theta(\mathbf{x} | M_t^{\text{mem}})$. In other words, the rate of intelligence growth in Figure 12.1 is higher when the new measurement result is more unexpected. This observation identifies scientist agents

[859] as a special type of curiosity-driven agent [869], prioritizing exploration over exploitation to expand the frontiers of knowledge for deeper understanding of nature. Unlike agents that leverage existing knowledge to achieve predefined objectives, curiosity-driven agents can learn without extrinsic rewards [387, 870] (see Section 5.3 for details), enabling discoveries beyond human-planned search spaces and revealing knowledge in unexplored domains. This potential also underscores the importance of equipping curiosity-driven agents with fundamental perception and action tools that can be transferred to explore new knowledge domains.

12.1.3 Intelligence Evolution Strategies

The strategy for expanding known information determines how quickly an agent’s intelligence evolves. For a given knowledge base M_t^{mem} , the parameter θ can be optimized over a space of world models Θ characterized by the architecture of M_t^{wm} . The optimal agent is the one that minimizes $D_K(\theta, M_t^{\text{mem}})$, thereby maximizing IQ_t^{agent} :

$$\theta_{K,t}^* \equiv \arg \sup_{\theta} IQ_t^{\text{agent}} = \arg \inf_{\theta} D_K(\theta, M_t^{\text{mem}}) \quad (12.6)$$

and

$$D_{K,\Theta}^{\min}(M_t^{\text{mem}}) \equiv D_K(\theta_{K,t}^*, M_t^{\text{mem}}) \quad (12.7)$$

Here, $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$ represents the minimum unknown after learning from M_t^{mem} for this family of models, quantifying the expressive limitations of Θ . As shown in Figure 12.1, $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$ forms the envelope of the family of functions $D_K(\theta, M_t^{\text{mem}})$, where θ ranges over Θ .

For a given model family Θ , $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$ measures the best possible prediction of residual unknowns in addressing the target scientific problem based on M_t^{mem} . In other words, the knowledge content in M_t^{mem} is captured by $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$. One can prove that $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$ is monotonically non-increasing as M_t^{mem} expands, since it forms the envelope of a family of non-increasing functions $D_K(\theta, M_t^{\text{mem}})$. This expansion process is tied to how the agent acts and gains information, driven by M_t^{wm} , which determines the optimal expansion and executes it through the action $a_t \in \mathcal{A}$ at time t (see Table 1.2).

During knowledge discovery, different strategies can be employed to expand M_t^{mem} . The optimal expansion strategy is the one that results in the steepest decrease of $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$. For instance, in Figure 12.1, we illustrate two strategies for expanding M_t^{mem} , denoted as $^1M_t^{\text{mem}}$ and $^2M_t^{\text{mem}}$. The first strategy, $^1M_t^{\text{mem}}$, represents random exploration, while the second, $^2M_t^{\text{mem}}$, follows a hypothesis-driven approach [871] in which the agent first formulates a hypothesis about the underlying mechanism of the target problem and then designs an experiment to justify or falsify this hypothesis [749]. In practice, experimentalists typically adopt the hypothesis-driven strategy because it enables them to guide the expansion of M_t^{mem} in a way that maximizes the reduction of $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$, subject to resource constraints. This approach is generally more efficient than random exploration for expanding M_t^{mem} , leading to $D_{K,\Theta}^{\min}(^2M_t^{\text{mem}})$ descending faster than $D_{K,\Theta}^{\min}(^1M_t^{\text{mem}})$.

In general, the knowledge discovery process proceeds iteratively, repeatedly optimizing the world model parameter θ to approach $\theta_{K,t}^*$ and expanding M_t^{mem} in a rational manner to accelerate the decrease of $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$. The ideal state is achieving epistemic completeness, i.e., $D_{K,\Theta}^{\min}(M_t^{\text{mem}}) = 0$, meaning zero discrepancy between the agent’s prediction and the real-world phenomena. However, for a specific agent, a discovery bound may exist, where $D_{K,\Theta}^{\min}(M_t^{\text{mem}})$ approaches zero but remains positive. These discrepancies arise from practical constraints and the limitations of Θ , \mathcal{A} , and other design spaces of the agent [872]. Achieving a low discovery bound requires designing an adaptive world model architecture, an efficient knowledge expansion strategy, and a sufficient action space.

12.2 Agent-Knowledge Interactions

Typical forms of scientific knowledge include observational knowledge (e.g., experimental measurements, computational results), methodological knowledge (e.g., experimental methods, computational techniques, protocols), and theoretical knowledge (e.g., theories, laws, predictive models). These forms of knowledge can contribute to scientific understanding as long as they consist of data and information processed in a way that affects the probability distribution of unknown information $P_{\theta}(\mathbf{x}_U | M_t^{\text{mem}})$, reduces $D_K(\theta, M_t^{\text{mem}})$, and facilitates decision-making.

In principle, external scientific knowledge has been shown to be useful in improving agent performance in reasoning and decision-making [873, 874]. However, the scope of this survey lies in how agents can autonomously discover and utilize knowledge to enhance themselves. Scientific knowledge discovery workflows typically involve hypothesis generation, protocol planning, conducting experiments and computations, analyzing data, deriving implications, and

revising hypotheses—often as part of an iterative cycle. An agent that can perceive, learn, reason, and act has the potential to drive such workflows in an autonomous manner, for example by using application programming interfaces (APIs) to interact with physical instruments to acquire scientific knowledge and iteratively enhance its knowledge base (Figure 12.2). The agent will use the acquired knowledge to update its mental states M_t to make better decisions when interacting with the world \mathcal{W} . We will now highlight three scenarios where agents discover scientific knowledge and enhance themselves.

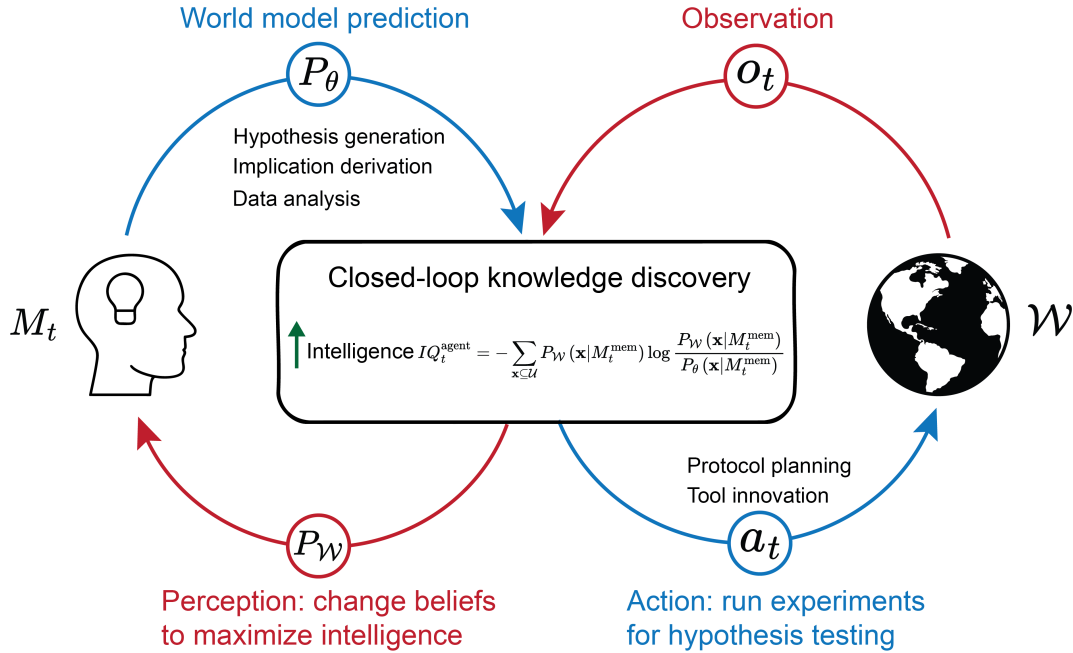


Figure 12.2: **Closed-loop knowledge discovery for sustainable self-evolution.** The agent aims to iteratively enhance its intelligence IQ_t^{agent} through hypothesis generation and testing, as well as through data analysis and implication derivation. When interacting with the physical world \mathcal{W} , the agent generates hypotheses as an explicitly or implicitly predicted distribution (P_{θ}) of unknown information, takes actions (a_t) for hypothesis testing, observes experimental results (o_t), and updates beliefs based on perception of the real-world distribution ($P_{\mathcal{W}}$). When not interacting with \mathcal{W} , the agent distills knowledge from existing data and premises, updating mental states M_t directly. Inspired by Figures 2.3 and 2.5 in [864].

12.2.1 Hypothesis Generation and Testing

Hypothesis generation and testing (Figure 12.2) is a critical application of agents in autonomous scientific discovery, as it has the potential to enable outside-the-box innovations [749]. In essence, hypothesis generation is the formation of potential rules that govern data distribution—ranging from single observations to large datasets—pertaining to unobserved scientific phenomena. According to Sir Karl Popper, a scientific hypothesis must be falsifiable [875, 876]; in this discussion, we define a hypothesis that survives falsification as a *justified true hypothesis* [877, 860]. Typically, scientists test hypotheses by conducting experiments to either justify or falsify them. A hypothesis is considered more valuable if it is broad enough to explain a wide range of data and is highly likely to be true.

To tackle a scientific problem, the agent formulates one or a small number of high-value hypotheses based on its mental state M_t , which contains only incomplete information about the partially observable world \mathcal{W} . After testing through experiments or computations, a *justified true hypothesis* becomes instructive knowledge, expanding M_t^{mem} in a way that rapidly minimizes $D_{K, \Theta}^{\min}(M_t^{\text{mem}})$. Hence, generating and testing high-value hypotheses can quickly promote knowledge discovery and increase IQ_t^{agent} . In this scenario, the agent employs the learning function, L , to process observations from hypothesis testing, o_t , into knowledge and update its mental states M_t .

Generating physically meaningful hypotheses is a key step. The agent typically uses LLMs along with collaborative architectures and domain knowledge for hypothesis generation [878]. Si et al. [742] conducted a large-scale human study involving over 100 NLP researchers, and found that LLM-generated ideas were rated as more novel ($p < 0.05$) than human expert ideas, albeit slightly weaker in feasibility. Ghafarollahi et al. [743] developed SciAgents, which generates

and refines materials science hypotheses to elucidate underlying mechanisms, design principles, and unexpected properties of biologically inspired materials. Based on large-scale ontological knowledge graphs, SciAgents samples a viable path between concepts of interest, formulates a pertinent hypothesis, and expands it into a full research proposal with detailed hypothesis-testing methods and criteria. It employs two dedicated agents to review, critique, and improve the proposed hypothesis, but does not include the step of hypothesis testing through actual experiments. Similarly, Su et al. [879] and Baek et al. [880] proposed leveraging teamwork—such as collaborative discussions and agent critics—to produce novel and effective scientific hypotheses. In addition, Gower et al. [881] introduced LGEM⁺, which utilizes a first-order logic framework to describe biochemical pathways and generate 2,094 unique candidate hypotheses for the automated abductive improvement of genome-scale metabolic models in the yeast *S. cerevisiae*.

Hypotheses only become knowledge after being justified through computational or experimental observations.

Lu et al. [745] introduced the AI Scientist, a system designed for fully automated scientific discovery. The AI Scientist can conduct research independently and communicate its findings, as demonstrated in three machine learning subfields—diffusion modeling, transformer-based language modeling, and learning dynamics. It generates original research ideas, writes code, performs computational experiments, visualizes results, drafts complete scientific papers, and even simulates a peer review process for evaluation. For instance, it proposed the hypothesis that “adaptive dual-scale denoising can improve diffusion models by balancing global structure and local details in generated samples,” which was justified through image generation tests on four 2D datasets. Similarly, Schmidgall et al. [746] developed the Agent Laboratory to autonomously carry out the entire research process, including literature review, computational experimentation, and report writing. They evaluated Agent Laboratory’s capability for knowledge discovery by addressing five research questions in computer vision and natural language processing, achieving an average human-evaluated experiment quality score of 3.2 out of 5. In addition, Tiukova et al. [744] developed Genesis, an automated system capable of controlling one thousand μ -bioreactors, performing mass spectrometry characterization, accessing a structured domain information database, and applying experimental observations to improve systems biology models. Genesis can initiate and execute 1,000 hypothesis-driven closed-loop experimental cycles per day. Using a similar approach, the Genesis team has advanced the yeast (*S. cerevisiae*) diauxic shift model, outperforming the previous best and expanding its knowledge by 92 genes (+45%) and 1,048 interactions (+147%) [882]. This knowledge also advances our understanding of cancer, the immune system, and aging. Similarly, Gottweis et al. [749] introduced the AI co-scientist, which autonomously generates and refines novel research hypotheses, with *in vitro* validation in three biomedical areas: drug repurposing, novel target discovery, and mechanisms of bacterial evolution and antimicrobial resistance.

Discovered knowledge enhances the agent’s mental states, such as M_t^{mem} , M_t^{wm} , and M_t^{rew} . Tang et al. [747] developed ChemAgent, which improves chemical reasoning through a dynamic, self-updating memory, M_t^{mem} . ChemAgent proposes hypothetical answers to chemistry questions in a development dataset, evaluates them against the ground truth, and simulates the hypothesis-testing process used in real-world research. Correct answers are then stored as knowledge in its memory to support future chemistry question answering. This self-updating memory resulted in performance gains of up to 46% (with GPT-4) when ChemAgent was applied to four chemical reasoning datasets from SciBench [883]. Wang et al. [884] introduced Molecular Language-Enhanced Evolutionary Optimization (MOLLEO), which iteratively proposes hypotheses for modifying candidate drug molecules in M_t^{mem} , evaluates their drug-likeness and activity, and updates the candidates in M_t^{mem} to enhance drug discovery. Similarly, Jia et al. [885] developed LLMatDesign, which employs hypothesis-guided structure generation and a self-updating M_t^{mem} to design inorganic photovoltaic materials, whose ideality is defined by matching the target band gap and having the most negative formation energy.

Sim et al. [748] introduced ChemOS 2.0, which orchestrates closed-loop operations in chemical self-driving laboratories (SDLs). ChemOS 2.0 integrates *ab initio* calculations, experimental orchestration, and statistical algorithms for the autonomous discovery of high-performance materials. A case study on discovering organic laser molecules demonstrates its capabilities. It employs a Bayesian optimizer, Altas, as its world model M_t^{wm} to predict the optical properties of hypothetical molecules—specifically Bis[(N-carbazole)styryl]biphenyl (BSBCz) derivatives—including gain cross section and spectral grain factor. Based on these predictions, ChemOS 2.0 recommends molecules with a higher probability of success in the experimental campaign. It then utilizes an optical characterization platform and the AiiDA software package to measure and simulate the properties of test molecules. The results are used to update M_t^{wm} , improving the accuracy of future experimental predictions.

Hysmith et al. [886] published a perspective highlighting the crucial role of reward function design in developing forward-looking workflows for SDLs. Agents can be highly effective at solving POMDP problems in simulated environments, such as computer games or simulations, but often struggle with real-world applications. A well-defined reward function is essential for iterative self-evolution. However, in many real-world scientific research problems, reward functions are ill-defined or absent at the end of experimental campaigns due to the lack of direct measurements,

the complexity of experimental results, and the need to balance multiple objectives. The discovery of new knowledge can serve as a valuable resource for refining M_t^{rew} , guiding hypothesis exploration and experimental data collection.

12.2.2 Protocol Planning and Tool Innovation

The capability to plan experimental protocols and optimize tool usage enables the agent to solve complex scientific puzzles within the autonomous discovery loop. As introduced in Section 9.4, the agent can systematically evaluate and refine its approach to selecting, invoking, and integrating available tools—and even develop new tools tailored to specific task requirements. While optimized protocols and tool usage do not directly reduce $D_K(\theta, M_t^{\text{mem}})$, they enhance execution efficiency and effectiveness in refining the probability distribution of unknown information, $P_\theta(\mathbf{x}_U | M_t^{\text{mem}})$, thereby accelerating knowledge discovery. In this scenario, the agent leverages the reasoning function R to translate its evolving mental states M_t , continuously updated with new knowledge, into real-world actions a_t for more effective and faster hypothesis testing (Figure 12.2).

Scheduling and orchestrating the selection and recombination of existing tools is critical. Scientific experiments typically depend on diverse instruments for analyzing reaction products, with decisions rarely rely on just one measurement. Effectively utilizing necessary instruments without wasting resources and time requires the agent to learn to use tools in an integrated and adaptive manner. Dai et al. [750] designed a modular workflow that integrates mobile robots, an automated synthesis platform, and various characterization instruments for autonomous discovery. They exemplified this system across three domains: structural diversification chemistry, supramolecular host-guest chemistry, and photochemical synthesis. The mobile robot follows a synthesis-analysis-decision cycle to mimic human experimental strategies, autonomously determining subsequent workflow steps. It selects appropriate instruments, such as the Chemspeed ISynth platform for synthesis, a liquid chromatography-mass spectrometer (UPLC-MS) for measuring mass spectra corresponding to chemical peak signals, and a benchtop nuclear magnetic resonance spectrometer (NMR) for tracking chemical transformations from starting materials to products.

Beyond individual laboratories, tool orchestration is essential for delocalized and asynchronous scientific discovery. Strieth-Kalthoff et al. [751] demonstrated a closed-loop integration of five materials science laboratories across three continents, advancing delocalized and democratized scientific discovery. These five laboratories have varying strengths—for example, the University of British Columbia specializes in continuous preferential crystallization, while Kyushu University excels in thin film fabrication and characterization. Strieth-Kalthoff et al. employed a cloud-based experiment planner to continuously learn from the incoming data and effectively prioritize informative experiments across the five laboratories, resulting in the discovery of 21 new state-of-the-art materials for organic solid-state lasers.

Moreover, the agent can optimize existing tools and even create new ones to enhance its capabilities. Swanson et al. [752] developed the Virtual Lab, an AI-driven research environment that facilitated the design and experimental validation of new SARS-CoV-2 nanobodies. Within the Virtual Lab, AI agents conduct scientific discussion in team meetings and execute specialized tasks in individual sessions. One key agenda for the agents was developing tools to aid in the design of nanobody binders [887], including: (1) a sequence analysis tool that ranks candidate point mutations using log-likelihood ratios from the ESM protein language model [888]; (2) a structure evaluation tool that extracts interface pLDDT scores from AlphaFold-Multimer predictions [889], offering a proxy for antibody-antigen binding affinity; and (3) an energy estimation tool built on Rosetta [890] to quantify binding strength between nanobody variants and the spike protein. These agent-generated tools enabled the Virtual Lab to discover two novel nanobodies with enhanced binding to the JN.1 or KP.3 SARS-CoV-2 variants, while preserving strong affinity for the ancestral viral spike protein.

12.2.3 Data Analysis and Implication Derivation

Although most knowledge discovery processes rely on generating hypotheses and testing them in the real world—where observations o_t are essential—a significant portion of knowledge can be derived purely through internal actions such as iterative reasoning and deep thinking, which are common in theoretical disciplines. For example, all theorems in Euclidean geometry can be deduced from just five axioms, but these theorems do not explicitly exist in the mental state before they are derived. Given all necessary premises, such as Euclid’s five postulates, the true probability of a hypothesis may remain elusive. However, using deductive and inductive reasoning to draw implications from known premises and data can help either justify or falsify hypotheses, thus reducing $D_K(\theta, M_t^{\text{mem}})$ and enhancing IQ_t^{agent} (Figure 12.2). In this scenario, the agent employs the cognition function C to use prior mental states M_{t-1} and internal actions a_t to derive new knowledge and update mental states to M_t .

Deductive reasoning enables knowledge derivation through logic. Trinh et al. [753] developed AlphaGeometry for the forward deduction of new mathematical theorems based on existing theorems in Euclidean plane geometry. AlphaGeometry employs a neural language model to construct auxiliary points in plane geometry problems and

integrates specialized symbolic engines to exhaustively deduce new true statements, thereby expanding the joint closure of known truths. By leveraging this expanded closure, it alternates between auxiliary constructions and symbolic reasoning engines to uncover further implications. AlphaGeometry demonstrated remarkable performance on a test set of 30 recent Olympiad-level problems, solving 25—more than double the 10 problems solved by the previous best method—and coming close to the level of an average International Mathematical Olympiad (IMO) gold medalist.

Inductive reasoning enables knowledge derivation through pattern recognition and statistical learning. Liu et al. [754] introduced the Team of AI-made Scientists (TAIS) to simulate the role of a data scientist for streamlined data analysis. TAIS decomposes a complex data analysis problem into different computational tasks, including coding, self-critique, and regression analysis, to extract meaningful insights from complex datasets. When applied to identifying disease-predictive genes, TAIS achieved an overall success rate of 45.73% on a benchmark dataset containing 457 genetic questions. Ideally, the extracted insights should be logically sound; otherwise, they must be discarded to ensure only accurate findings are safely integrated into mental states. However, limitation in data coverage and the implementation of analysis algorithms may lead to hallucinated insights, underscoring the need for reliable data analyzers and reasoning tools to prevent over-analysis.

12.3 Technological Readiness and Challenges

The self-evolution of agents, which in turn drives the advancement of human knowledge, is promised by their early success in the innovation cycle. This cycle involves generating meaningful hypotheses, designing real-time testing protocols, coordinating various experimental and computational tools, analyzing data, deriving implications, and engaging in self-reflection. However, achieving fully autonomous self-evolution remains a significant challenge, given the current technology readiness levels (TRLs) of three fundamental capabilities: real-world interaction, complex reasoning, and the integration of prior knowledge. Further technological progress is required to improve the cycle of self-driven innovation.

12.3.1 Real-World Interaction Challenges

Agents interact with the real world primarily through application programming interfaces (APIs). While numerous demonstrations [891] have shown their strong capability to use various APIs, a significant bottleneck in autonomous knowledge discovery remains: the lack of APIs that allow agents to directly execute tasks in a physical laboratory. Physical APIs—interfaces that enable direct control of lab equipment—are far less abundant than computational APIs due to the significant investment of time, expertise, and cost required to develop them. Although existing autonomous laboratories have shown promise, they remain in an early developmental stage (typically TRL 4–6), where straightforward replication or scale-up is challenging. Consequently, building further systems or broadening their application across additional scientific domains still requires substantial customization to address domain-specific needs, along with specialized expertise.

Two key tasks are essential for enabling real-world interaction: *operating lab devices* and *transferring samples between devices*. Seamless integration of physical hardware and experimental samples is crucial to maintaining uninterrupted workflows. However, most experimental instruments are originally designed for human operation. Making them accessible to agents requires extensive efforts across multiple disciplines, including robotics, electrical engineering, mechanical engineering, and software programming. The rising prominence of SDLs is catalyzing the transformation of human-operated devices into agent-accessible systems through APIs. In autonomous labs conducting complex experiments, two parallel and often complementary approaches are commonly adopted to integrate hardware with agentic systems. Both approaches are modular, reconfigurable, and valuable, yet they require ongoing, dedicated development.

Approach 1: API Integration via Direct Device Adaptation. This approach involves equipping individual devices with dedicated mechanical adaptations and I/O controllers, enabling them to receive and execute commands from a central control PC. For example, to achieve solid-state synthesis and structural characterization of inorganic materials, A-lab has implemented 16 types of devices to automate experimental tasks such as powder dosing, heating, and diffraction [892]. This approach allows laboratories to function as fully integrated entities by maximizing device utilization, optimizing space and resources, and enabling bespoke tools. However, it is costly, time-consuming, and requires expert knowledge to prototype or retrofit devices for automation. Large language models (LLMs) have been applied to facilitate access to diverse tools, as illustrated by CACTUS, a Chemistry Agent Connecting Tool-Usage to Science [893].

A more accessible alternative for small teams is the *cloud lab* or *science factory* [894], where responsibility for device engineering shifts from individual laboratories to dedicated user facilities or commercial service providers. For

instance, Boiko et al. [895] demonstrated an autonomous chemical research agent, Coscientist, capable of carrying out cross-coupling Suzuki and Sonogashira reactions using experimental setups at the Emerald Cloud Lab [896]. However, cloud labs offer only a fixed set of pre-built devices optimized for common procedures, posing potential challenges for researchers whose experiments require equipment customization, as integrating non-standard tools may involve a lengthy process of negotiation and development.

Approach 2: Robotic Operation of Experimental Devices. This approach involves using mobile robots or robotic arms to operate existing devices and transfer samples. In many cases, robots can interact with instruments without modification, apart from minor adjustments such as adding specialized actuators, grippers, or holders. For example, Dai et al. [750] employed mobile robots to explore synthetic chemistry. In their autonomous laboratory, mobile robots enable physical linkages between synthesis and analysis devices that are spatially separated, automating sample transportation and handling. In principle, the robots can perform all actions human researchers require in the laboratory. However, current robotic systems still rely on human pre-programming to map the lab layout, define movement trajectories, and register device positions. Handling unexpected or adaptive situations remains a challenge, as pre-programming cannot anticipate every possible state of an experimental setup. Real-time learning and adaptive manipulation are active areas of research that require further technological advancements. In the long term, embodied AI [897] is expected to enhance robotic learning, allowing agents to quickly adapt to new environments and tools.

The two approaches can be combined. For example, Vescovi et al. [894] define a modular laboratory robotics architecture that allows for translating high-level commands into specific operations for a variety of different robotic apparatus and laboratory equipment, and for linking robotic apparatus with other elements of an AI-driven discovery architecture, such as high-performance computing [898]. This architecture has been used to automate experiments in both the biological and physical sciences [899]. Similarly, Fernando et al. [900] integrate a Robotic Operating System 2 (ROS2) compatible robot into the Bluesky experimental orchestration framework. Lo et al. [901] argue for the development and integration of low-cost “frugal twins” of more expensive equipment to facilitate experimentation and democratize access.

12.3.2 Complex Reasoning Challenges

A fundamental philosophical question is whether agents, often powered by LLMs, can truly perform reasoning. By definition, language models generate outputs by predicting the next token, a mechanism fundamentally different from human reasoning. From an outcome-driven perspective, these input-output systems exhibit reasoning ability phenomenologically, as they produce meaningful outputs compared to a reference system generating arbitrary responses [902]. However, regardless of the perspective taken, this capability remains imperfect—particularly when handling complex logical and numerical problems, which are crucial for scientific knowledge discovery.

Agents and LLMs struggle with hard reasoning tasks. Glazer et al. [903] introduced FrontierMath, a benchmark comprising hundreds of original and challenging mathematics problems covering most major branches of modern mathematics. Evaluation of state-of-the-art LLM-driven agents—including o1-preview (OpenAI), o1-mini (OpenAI), GPT-4o (OpenAI, 2024-08-06 version), Claude 3.5 Sonnet (Anthropic, 2024-10-22 version), Grok 2 Beta (XAI), and Gemini 1.5 Pro 002 (Google DeepMind)—revealed that no model achieved even a 2% success rate on the full benchmark. Chen et al. [873] presented ScienceAgentBench, a benchmark designed to evaluate language agents in data-driven scientific discovery. Among 102 tasks derived from 44 peer-reviewed publications across four disciplines, OpenAI o1 successfully solved only 42.2% of them. Chollet [865] proposed the Abstraction and Reasoning Challenge (ARC) to assess LLMs’ ability to perform abstract inductive reasoning without relying on memorization or external knowledge. Even with careful prompting, GPT-4o correctly solved only 19% of the tasks, far below the ~75% average human performance [904, 905]. Zhu et al. [906] suggested a four-level classification of AI intelligence, including L1 (arbitrating isputes), L2 (auditing a review), L3 (reviewing a paper), and L4 (authoring a paper). They classify the current state-of-the-art LLM-driven agents as approaching L2-level capabilities. To enhance agents’ reasoning abilities, researchers have introduced techniques such as chain-of-thought [907], tree-of-thoughts [72], and [70]. Although new methods continue to emerge, as discussed in Section 2.2, further advancements in reasoning capacity remain crucial for achieving reliable causal inference in scientific research.

Agents and LLMs also struggle with quantitative and symbolic problems. For example, GPT-4 and GPT-3.5 often struggle with reliably performing complex arithmetic such as multiplying $12,345 \times 98,765$, or translating IUPAC chemical names into accurate molecular graphs [908, 697]. A common approach to overcoming these limitations is to use external tools rather than relying on the LLM itself for reasoning. In mathematical problem-solving, for example, tools like symbolic solvers are preferred over direct LLM inference [753]. However, this mitigation does not resolve the intrinsic deficiency in numerical understanding, which poses a potential risk to scientific reasoning. Moreover, Yu et al. [909] found that tool-augmented LLMs do not consistently outperform base LLMs without tools in chemistry problem-solving. For instance, for *specialized* chemistry tasks, such as synthesis prediction, augmenting

LLMs with specialized tools can boost the performance substantially; however, tool augmentation is less effective for *general* chemistry questions, such as those in exams, where no specific tools can directly solve a given question. In these scenarios, an agent’s ability to reason correctly by using multiple pieces of chemistry knowledge becomes more important.

The preceding discussion emphasizes the importance of developing robust methodologies for evaluating AI agents as scientific research assistants, a topic discussed at length by Cappello et al. [910].

12.3.3 Challenges in Integrating Prior Knowledge

Prior knowledge is a crucial factor for higher intelligence. As discussed in Section 12.1, the agent’s prior knowledge, M_t^{mem} , helps decrease $D_K(\theta, M_t^{\text{mem}})$ and increase the agent’s intelligence, IQ_t^{agent} . Human-led scientific discoveries frequently achieve breakthroughs with relatively small datasets, thanks to the vast prior knowledge humans possess. The start-of-the-art LLMs that power autonomous agents are trained on nearly all publicly available textual data, including websites, books, and other sources, thereby encompassing most common knowledge as well as publicly accessible specialized knowledge. However, achieving an agent that can seamlessly integrate all existing human knowledge remains a significant challenge.

At least three types of knowledge sources may not be included in LLM pre-training: (1) Paywalled or unpublished knowledge, including non-open-access publications, industry-specific data, and failed experiments [911]. They are often not accessible to public models despite their potential value in refining domain-specific insights. (2) Empirical knowledge. Heuristic decisions by experts are often effective, particularly in scenarios where no existing data is available for a new problem. However, large amounts of expert heuristics are typically not accessible as textual data. (3) Contextual or situational knowledge. Knowledge related to real-world conditions, such as safety protocols in chemical reactions or equipment handling, is often absent from pre-trained models but is essential for practical applications.

Additionally, integrating diverse knowledge sources presents challenges in reconciling conflicting information. For example, OpenAI’s Deep Research [912] actively gathers online information and performs multi-step reasoning, achieving state-of-the-art performance on Humanity’s Last Exam and the GAIA benchmark. However, it still struggles to distinguish between authoritative information and rumors and exhibits limitations in confidence calibration, often misrepresenting its level of certainty [912]. Establishing a system to assess the levels of evidence [913] of different knowledge fragments—such as quantifying reliability and verifying references—may be necessary for effective knowledge fusion.